Raimund Ubar

Digitaalsüsteemide diagnostika

I

Diagnostiline modelleerimine

TTÜ
KIRJASTUS

Selle õppevahendi väljaandmist
Toetas Eesti Infotehnoloogia Sihtasutus
Tiigriülikooli projekti kaudu

Kaanekujundus: Tiina Ubar Sauter

Sisukord

Eessõna

“Eksimine on inimlik.
Et aga tõeliselt kõik ära rikkuda,
on vaja arvutit”
(insenerifolkloor)

Viimase 10 aastaga on inimkond rohkem teadmisi produtseerinud kui kogu eelneva tsivilisatsiooni vältel. Selle tulemusena on tekkinud meeletu kiirusega arenev tehismaailm, kus ühelt poolt, materiaalsed artifaktid on suurendanud inimese tugevust ja kiirust, ning teiselt poolt, kognitiivsed artifaktid (arvutid) on teinud teda targemaks ja kavalamaks.

Arvutite võimsus kahekordistub pooleteise aastaga. Aastast 1950 kuni tänaseni on see kasvanud 10 miljardit korda. See on palju rohkem, kui see võimsuse kasv, mille tõi kaasa üleminek dünamiidilt vesinikupommile. Hiilivalt pealetungiv infotehnoloogiline *ubikvism* (nähtamatud arvutid kõikjal ja ei kusagil – *ubiquitous computing*) on muutmas ulmeraamatute stsenaariumid tõelisuseks.

Tehnoloogia keerukus kasvab, aga keerukus kätkeb endas ootamatusi ja mittekavandatut. Elu on täis ohte, aga võrku ühendatud arvutid, mis juhivad ja kontrollivad tehismaailma, on täis veelgi suuremaid ohte: arvutirikete põhjustatud lennukatastroofid, õnnetused keemiatööstuses, tuumajaamades… Tehissüsteemide ohutus ja turvalisus on muutumas sageli olulisemaks kui niisuguste süsteemide täiuslikkus.

Esimest arvutiviga maailmas dateeritakse aastaga 1945, kui koiliblikas sattus Harwardis arvuti Mark II kahe relee vahele. 1991. a. Lahesõjas tabab Iraaki suunatud rakett USA enda raketibaasi, kuna ümardamisvea tõttu arvutas kompuuter valesti raketi trajektoori. 1995. a. plahvatab Ariane-5 juba 40 sek. pärast starti tarkvaravea tõttu inertsiaalsüsteemis. 1996. a. saadab Intel turule mikroprotsessori, mille jagamistehe töötab riistvara vea tõttu valesti. 1998. a. USA kogukahjum arvutivigadest oli 11 miljardit dollarit. 43% firmadest, keda on tabanud andmekatastroof arvutisüsteemis, pankrotistub.

Üha suurenev sõltuvus tehnoloogiast teeb inimese kergesti haavatavaks ja abituks olukordades, kus tehnika hakkab tõrkuma. Harilikult märkamegi seda sõltuvust alles siis, kui saabub õnnetus, milleks me valmis pole olnud.

Möödunud sajandi keskel eeskätt lennuki- ja raketitööstuse vajadusel kujunes välja inseneriteadus, mida nimetatakse *tehniliseks diagnostikaks*. Eriti suure arenguhoo sai see teadus arvutite ilmumisel tehismaailma, mille põhjuseks oli tehissüsteemide s.t. arvutite keerukuse kiire kasv. Paradoksaalne on see, et keerukuse teooria loodi digitaalse tehismaailma jaoks, mis oma olemuselt on lõplik, aga mitte analoogmaailma jaoks, mille dimensiooniks on lõpmatus.

Submikron-tehnoloogia ning juba käivitunud investeeringud nanotehnoloogiasse elektroonikas võimaldavad inseneridel luua üha keerukamaid mikrokiipe - integraalskeeme. Arvutid ja arvutivõrgud ühesainsas mikroskeemis on saanud tavapäraseks kõrgtehnoloogiliseks produktiks tänapäeval.

Mida keerukam on aga süsteem, seda kindlamini kehtib Murphy seadus: *kui miski võib untsu minna, siis ta ka läheb*. Keerukus tähendab rohkete vigade võimalusi nii süsteemide projekteerimisel, valmistamisel kui ka nende kasutamisel. Seega, mida keerukam on süsteem, seda olulisemaks muutuvad süsteemi testimine, diagnostika ja veakindlus. Kulutused elektroonikasüsteemide testimisele ulatuvad sageli kuni 70%-ni (!) kõikidest tootmiskuludest tööstuses. Kuidas projekteerida süsteeme nii, et nad oleksid kergemini (odavamalt) ja paremini testitavad – on uus inseneriprobleem süsteemitehnikas.

Elektroonika-, arvuti- ja süsteemiinseneride hariduse oluliseks komponendiks on tänapäeval saanud teadmised ja oskused skeemide, seadmete ja süsteemide testimise ning diagnostika alal. Süsteemide usaldatavuse ja veakindlusega seotud distsipliine on hakatud õpetama tehnikaülikoolides üha laialdasemamalt ja põhjalikumalt.

Laiemaski mõttes tähendab tehniline diagnostika kui õppeaine intellektuaalset harjutust põhjuste ja tagajärgede mõtestamisel ning omab seetõttu küllaltki olulist didaktilist tähendust insenerihariduse kujundamisel. *Diagnostika* on aine, mis õpetab formuleerima õigeid küsimusi, õpetab planeerima eksperimente ning õpetab analüüsima eksperimentide tulemusi. Tehissüsteemide maailm on rikas eri valdkondade iseärasuste poolest, aga ühiseks siduvaks jooneks on süsteemide loogikaline keerukus. Seetõttu ongi just digitaalsüsteemid tehismaailma kõige iseloomulikumaks esindajaks tehnilise diagnostika probleemide uurimisel ja mõistmisel.

Viimastel aastatel on ilmunud rohkesti võõrkeelset õppe- teadus- ja teatmealast kirjandust elektronskeemide, arvutite ja digitaalsüsteemide testimise ning diagnostika kohta.

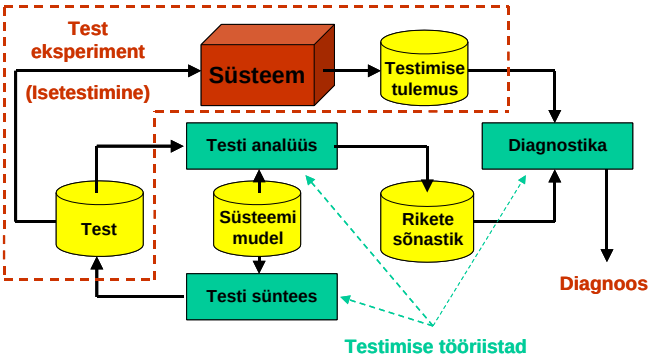
Käesoleva raamatu eesmärgiks on täita lünk eestikeelse kirjanduse puudumise osas nimetatud valdkonnas.

Raamat on plaanis avaldada mitmeosalisena. Esimeses osas käsitletakse digitaalsüsteemide diagnostilise modelleerimise teoreetilisi aluseid, mille eesmärgiks on kujundada baas diagnostikaprobleemide matemaatiliseks uurimiseks ning analüüsiks, nende lahendamise algoritmiseerimiseks ja automatiseerimiseks. Järgmistes osades vaadeldakse diagnostikatestide sünteesi ja analüüsi probleeme, testprogrammide automaatse genereerimise meetodeid, rikete simuleerimist ja diagnoosi ehk rikete asukoha lokaliseerimist, süsteemide projekteerimise ja diagnostika omavahelisi seoseid, seda, kuidas luua hõlpsamini testitavaid ja diagnoositavaid süsteeme, kuidas projekteerida iseennast testivaid seadmeid ja kuidas tagada süsteemide usaldatavust ning veakindlust.

Sissejuhatus

Digitaalsüsteemide *tehnilise diagnostika* eesmärgiks on testimise abil kindlaks teha, kas süsteem on töökorras ja kui ta ei ole töökorras, siis lokaliseerida rikete asukoht nende kõrvaldamiseks. Laiemas mõttes tähendab *diagnostika* nii *testimist* kui rikete lokaliseerimist, kitsamas mõttes ainult *rikete lokaliseerimist* ehk diagnoosi.

Diagnostika tähtsamaid omavahel seotud ülesandeid ja nende lahendamiseks kasutatavaid tööriistu esitab ülevaatlikult joonis 1. Keskseks objektiks on siin testimisele ja diagnoosimisele kuuluv *tehniline süsteem*. *Testeksperiment* selle süsteemiga võidakse läbi viia kas spetsiaalse välise testimisseadme (automaatse *testri*) abil või süsteemi enda ressursside poolt, mida nimetatakse *isetestimiseks*.

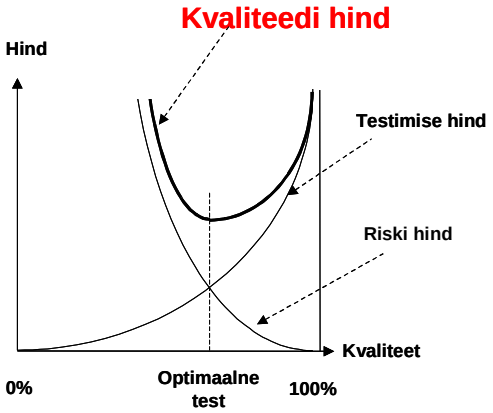


Joonis 1. *Testimise tööriistad*

Mõlemal testimise juhul on vaja rakendada *testi* ehk *testvektorite* jada, mis on eelnevalt genereeritud ja hoitakse testri mälus või mida genereeritakse jooksvalt (ehk *on-line*) spetsiaalsete süsteemi sisseehitatud

vahendite poolt isetestimise käigus. *Testi kvaliteet* (avastatavate rikete protsent kõigi võimalike rikete suhtes) tehakse kindlaks *testi analüüsi* tööriistadega, milleks on harilikult tarkvaralised testide analüsaatorid ehk *rikete simulaatorid*. Viimaste abil konstrueeritakse ka *rikete sõnastik*, mida saab tarvitada tehnilise diagnoosi panemiseks. *Testide sünteesiks* kasutatakse väga keerulisi tarkvaratööriistu – testide generaatoreid.

Mida tähendab *testigeneraator*? See on programm, mis analüüsib sadadest tuhandetest loogikaelementidest koosnevat elektronskeemi (õigemini, tema mudelit), määrab kindlaks kõik selles skeemis esineda võivad *rikked* – lühised ja katkestused elementide sees ning ühenduste vahel – ja seejärel hakkab iga sellise rikke jaoks otsima niisugust signaalide kogumit (*testi*), mille rakendamisel skeemi sisenditele avastataks nimetatud rike juhul, kui see skeemis ette tuleks. Kui skeem koosneks mõnest tuhandest elemendist, siis sünteesiksid parimad generaatorid selle testi mõne minutiga. Mõnekümne tuhande elemendi puhul võib kuluda aega juba tunde, ja edasi kasvava keerukuse puhul suureneks vajalik aeg juba eksponentsiaalselt. Reaalsed digitaalsüsteemid koosnevad aga miljonitest elementidest. Kuidas siis genereerida niisugustele süsteemidele teste, jäädes seejuures realistliku aja piiridesse?



Joonis 2. *Testimise ja riski maksumuse vahekord*

Tehnilise diagnostika filosoofiliseks põhiküsimuseks ongi – *kui palju testida* uurimise all olevat süsteemi olemaks kindel, et süsteem on töökorras. Teiseks tähtsaks, aga rohkem juba tehniliseks põhiküsimuseks on: kus asub rike, kui on testimise teel juba kindlaks tehtud, et süsteem ei

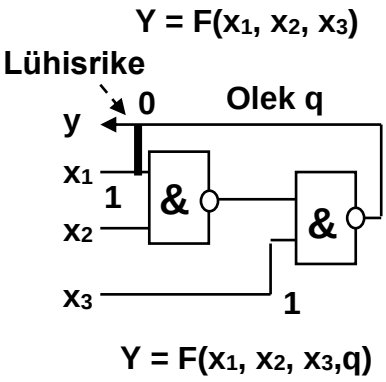
tööta. Mõlema probleemi lahendus koosneb kahest poolest – testide sünteesist (genereerimisest) ja testide analüüsist.

Testimine on tegelikult kvaliteedi ja selle tagamiseks vajalike kulutuste ehk raha dilemma. Mida rohkem ja põhjalikumalt testime, seda kindlamalt garanteerime süsteemi kvaliteeti (loe siinkohal: *usaldusväärsust*). Aga testimine tähendab kulutusi. Mida rohkem testime, seda rohkem kulutame aega ja ressursse ehk kokkuvõttes raha. Samas, kui ei testiks üldse või testiks liiga vähe, siis *riskiksime* vigade tekkimisega süsteemi töös võimalike avastamata jäänud rikete tõttu ja kõigest sellest tulenevate pahandustega, mida lõppkokkuvõttes trahvide, kompensatsioonide jms. näol tuleks lõpuks kinni maksta. Nii testimisel on hind kui ka *riskil* on hind. Kusagil on aga kompromiss, nagu näitavad joonisel 2 olevad kõverad.

Kvaliteedi hinna kõveral on miinimum, mida ilmselt tulekski taotleda. See miinimum tähendab sisuliselt parimat kulutuste ja riski vahekorda.

Probleem on vaid selles, et täpseid kõveraid ja parimat kompromissi kulutuste ja riski vahel on väga raske välja arvutada.

Esimeseks raskuseks oleks testi *100%-lise kvaliteedi* määrang. Kuidas kindlaks teha, et test annab 100%-lise garantii süsteemi töökorras oleku kohta? Üheks võimaluseks oleks läheneda probleemile "terve mõistuse" loogika abil: kui süsteem töötab, siis on ta ka töökorras. Aga kuidas kindlaks teha, et süsteem töötab?



Joonis 3. *Lühisrike digitaalskeemis*

Võtame näiteks 32-bitise *summaatori*. Selleks, et kindlaks teha, et summaator töötab õigesti, tuleks läbi summeerida

$$2^{64} = 18\,446\,700\,000\,000\,000\,000 \approx 10^{19}$$

arvupaari. Selle sooritamiseks kuluks 1GHz summeerimissageduse juures 3 sajandit (!). Aga isegi see poleks piisav (!).

Kujutleme *lühisriket* joonisel 3 näidatud skeemi ühe sisendi ja väljundi vahel. 3-sisendilise *kombinatsioonskeemi* $y = F(x_1,x_2,x_3)$ täielikuks testimiseks oleks vaja $2^3 = 8$ testvektorit. Aga lühise tõttu pole enam tegemist kombinatsioonskeemiga. Rikke tulemusena on tekkinud triger ehk *järjestikskeem* (lõplik automaat) olekuga q, mille väljundfunktsioonil $y = F(x_1,x_2,x_3,q)$ on mitte 3 vaid 4 argumenti. Seetõttu oleks täielikuks testimiseks vaja nüüd $2^4 = 16$ testvektorit. Pöördudes tagasi summaatori juurde, tähendaks ühe oleku lisandumine *testimisaja* kahekordistumist ehk 3 sajandi asemel 6 sajandit. Kuna võimalike sekventsiaalsete lühiste arv võib suuremas skeemis küllalt suur olla ja kuna arvesse tuleks võtta ka lühiste suvalisi kombinatsioone, siis näemegi, et kõigi testimiseks vajalike testvektorite arvu ülemine piir kipub praktiliselt lõpmatu suureks kujunema.

Väljapääs olukorrast oleks loobuda niisuguse *funktsionaalse testimise* käsitlusest ja asendada see *struktuurse* lähenemisega. Näiteks 32-bitist summaatorit võib kirjeldada kombinatsioonskeemina, mis realiseerib 32 loogikafunktsiooni paari

$$s_i = a_i \overline{b_i}\,\overline{c_i} \vee \overline{a_i} b_i \overline{c_i} \vee \overline{a_i}\,\overline{b_i} c_i \vee a_i b_i c_i$$

$$c_{i+1} = a_i b_i \vee a_i c_i \vee b_i c_i .$$

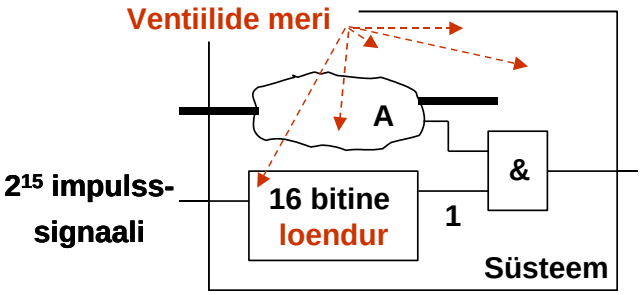
Kokku oleks niisuguses skeemis 32*3 sisendit, 32*18 sisendharu ja 32*7 ühendust ehk kokku 896 juhet. Igal juhtmel võib kujutleda kahte võimalikku *loogilist riket* “konstant 0” ja “konstant 1”, seega kõikide rikete koguarv oleks 1792. Iga niisuguse rikke avastamiseks oleks vaja leida üks testvektor ehk kahe summeeritava arvu paar. See oleks ühtlasi ka *testimise keerukuse* mõõduks, mis oleks juba päris realistlik. Edaspidi näeme (raamatu teises osas), et tegelikult piisaks suvalise sõnapikkusega *järjestikülekandega* summaatori testimiseks ka ainult 8-st arvupaarist.

On lihtne näha, et struktuurse ehk *riketepõhise* lähenemisviisi puhul ülesande keerukus kujuneb mõõtmatult väiksemaks kui funktsionaalse lähenemise puhul. Ometigi põrkame ka siin kohe raskustele: skeemis võivad ju põhimõtteliselt tekkida ka *mitmekordsed* rikked. On lihtne arvutada, et n-juhtmelises skeemis on kõigi võimalike kordsete rikete arv 3^n-1, ehk antud juhul 32-bitise summaatori puhul 3^{1792}-1. Iga juhe võib olla kolmes

võimalikus olekus: “konstant 0”, “konstant 1” või “korras”. Rikete loetlemisel ei tule arvesse situatsioon, kus kõik juhtmed on “korras”. Seega ka struktuurse lähenemise korral viib täiuslikkuse taotlemine meid keerukuse lõksu ja praktiliste lahenduste leidmiseks tuleb ülesannet “mõistlikult” lihtsustada. Üheks niisuguseks võimaluseks oleks eeldus, et skeemis võib alati olla korraga ainult üks rike. Sellise eelduse põhjenduseks on see, et kordsete rikete tekkimise tõenäosus on väike: kui ühe *rikke tõenäosus* oleks näiteks 1 tuhandik, siis kahe üheaegse rikke tõenäosuseks oleks 1 miljondik. Nii või teisiti on selline eeldus ikkagi kompromiss reaalsete testimisvõimaluste ja saavutatava testimiskvaliteedi vahel.

Kohe tulevad aga uued probleemid. Konstantsete rikete mudel *loogikaskeemide* tasemel ei ole piisavalt täpne esindamaks reaalseid füüsikalisi defekte *transistorskeemide* tasandil. Aga asendades analüüsi eesmärgil loogikaskeemid transistorskeemidega sattuksime uuesti keerukuse kiire kasvu umbteele.

Nii jõuamegi uue üldisema põhiküsimuse juurde digitaalsüsteemide diagnostikas: kuidas tagada süsteemide üha kasvava keerukuse tingimustes testimise kvaliteeti.

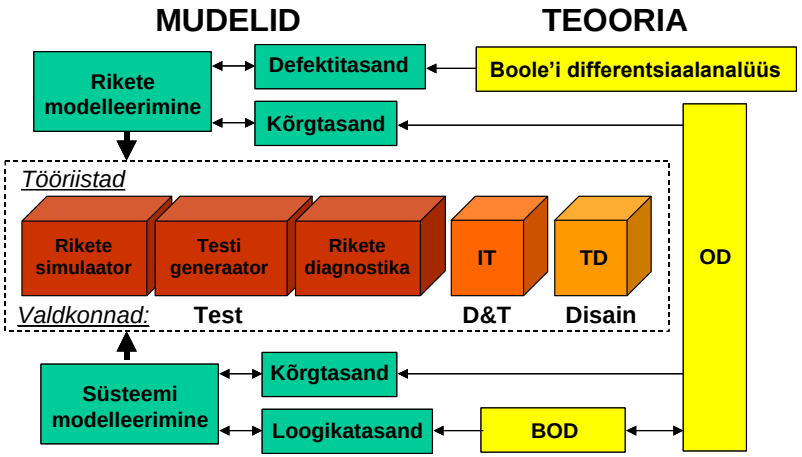


Joonis 4. *Hierarhiline süsteemi modelleerimine*

Keerukusega saaks “võidelda”, kui *süsteemi analüüsi* (testide sünteesi ja rikete simuleerimist) viia läbi kõrgematel tasanditel väiksema detailsusastmega mudelitel. Samal ajal, nagu eelpool toodust juba järeldada võis, kaotaksime niiviisi analüüsi täpsuses ega saa tagada testide kvaliteedi adekvaatset hindamist. Teiselt poolt, täpsust saaks tagada küll, kui viime testide analüüsi läbi *modelleerides* süsteemi *defektipõhiselt* võimalikult madalal näit. transistorskeemide tasandil. See aga suurendaks taas uurimisobjekti keerukust ega võimaldaks saavutada praktilisi tulemusi.

Väljapääsuks ummiksituatsioonist oleks *hierarhiline* lähenemisviis keerukate digitaalsüsteemide testimisel.

Joonisel 4 on esitatud *digitaalsüsteem*, mida võib käsitleda näiteks kas kui struktuuritut "*ventiilide merd*" ("*sea of gates*") või siis hierarhiliselt, kus üksikutele ventiilide piirkondadele on seatud vastavusse kõrgema tasandi funktsioonid. Näiteks ühele ventiilide grupile joonisel 4 on vastavusse seatud 16-bitise *loenduri* funktsioon. Oletame, et on vaja genereerida test ploki A võimalike rikete avastamiseks süsteemi väljundis. Selleks, et rikete tõttu tekkivad veasignaalid leviksid süsteemi väljundisse, kus neid avastada saaks, tuleks JA-elemendi alumisele sisendile saata signaal 1. Kui me ei teaks, et selle sisendiga ühenduses olev skeem on loendur ja seda skeemi tuleks käsitleda kui "ventiilide merd", oleks väga raske loogikameetodite abil tuletada niisugune sisendsignaalide jada, mis tekitaks meil JA-elemendi sisendis vajaliku signaali 1. Taoline tuletusprotsess võiks võtta ka väga võimsalt arvutilt aega minuteid kui mitte tunde. Teades, aga, et nimetatud ventiilide grupp on 16-järguline loendur, oleks ka ilma arvutusteta selge, et signaali 1 saamiseks loenduri väljundis on vaja tema sisendile rakendada 2^{15} impulsist koosnev signaalide jada.



Joonis 5. *Diagnostika probleemide üldplaan*

Hierarhiline lähenemisviis tähendab seda, et me osa süsteemi analüüsi viime läbi komponentide tasemel, mis on vähem keerukad, ja mis seetõttu võimaldavad töötada madalamatel ja detailsematel (ventiilide või isegi transistoride) tasanditel. Nii õnnestuks adekvaatselt käsitleda füüsikalisi

defekte ja leida tingimusi, kuidas neid aktiveerida, nii et nende mõjusid ka kõrgematel tasanditel jälgida saaks. Defektidest tingitud veasignaalide analüüs saaks seejärel toimuda juba pealiskaudsemaid kõrgema tasandi mudeleid kasutades.

Ülaltoodust tuleneb, et keerukate süsteemide diagnostikaprobleemide mõistmiseks ja nende lahendamiseks tuleb tunda mitmesuguseid süsteemide eri tasanditel töötavaid matemaatilisi teooriaid, mudeleid ja meetodeid.

Joonisel 5 on esitatud digitaalsüsteemide *diagnostika probleemide* üldplaan. Probleemid võib jagada kolme valdkonda, mis on omavahel küllaltki läbipõimunud: *testimine, disain ja test* ning *testikõlblik disain*. *Testi* valdkond ühendab endas testide sünteesi ja analüüsiga seotud probleeme, mida võiks eristada kui rikete simuleerimine, testide genereerimine ja diagnoos. *Disaini ja testi* (D&T) valdkonda kuuluvad küsimused, kuidas projekteerida (disainida) isetestivaid süsteeme (ITS). Disaini ja testi probleemid on siin väga läbipõimunud. *Testikõlblik disain* (TD) aga haarab probleeme, kuidas projekteerida süsteemi nii, et see oleks võimalikult hästi, kvaliteetselt ja kergesti testitav.

Probleemide lahendus seisneb vajalike tööriistade välja töötamises, valmis ehitamises ja efektiivses rakendamises. Tööriistade all mõtleme siin eritarkvara (rikete simulaatorid, testigeneraatorid, rikete diagnoosi ja lokaliseerimise tarkvara). Tööriistade väljatöötamise aluseks on meetodid ja algoritmid, mis töötavad süsteemi ehk diagnostikaobjekti matemaatilistel mudelitel. Hierarhilise lähenemisviisi realiseerimiseks peab meil olema nii madala kui ka kõrgtasandi mudeleid. Algoritmide väljatöötamiseks eri tasanditel ja eri mudelitel tuleb kasutada mitmesuguseid teooriaid.

Paljude võimalike teoreetiliste aparatuuride hulgast on käesolevas broshüüris põhjalikumaks käsitlemiseks välja valitud kaks: *Boole'i differentsiaalalgebra* ja *otsustusdiagrammid*, mis kokku võimaldavad anda ühtse ja kompaktse käsitluse põhilistest digitaalsüsteemide diagnostika probleemidest. Boole'i differentsiaalaparaat võimaldab siduda kaks põhilist diagnostikaprobleemi - testide süntees ja rikete diagnoos – ühte ainsasse Boole'i differentsiaalvõrrandisse. Boole'i tuletiste abil osutub võimalikuks ka defektide kujutamine füüsikatasandilt loogikatasandile, mis oleks esimeseks sammuks defektide hierarhilisel modelleerimisel. *Binaarsete otsustusdiagrammide* (BOD) abil õnnestub kompaktselt esitada põhilisi testide sünteesi ja analüüsiga seotud algoritme. Loobudes aga binaarsuse kitsendusest õnnestub neid algoritme otsustusdiagrammide (OD) üldjuhu abil üldistada digitaalsüsteemide kõrgematele tasanditele.

1. Digitaalsüsteemide diagnostiline modelleerimine

Digitaalsüsteemide *diagnostilise modelleerimise* eesmärgiks on analüüsida süsteemide käitumist erinevate rikete korral, seletada põhjusi, miks süsteem käitub konkreetses olukorras ebanormaalselt, genereerida teste erinevate rikete avastamiseks, analüüsida testide võimet avastada rikkeid ja lahendada palju muid spetsiifilisemaid süsteemide diagnostikaga seotud ülesandeid. Komplitseerituks muudab nimetatud probleemide lahendamise digitaalsüsteemide üha kasvav keerukus. Väljapääsuks olukorrast on hierarhiline lähenemisviis meetodil "*jaga ja valitse*", mis eeldab seda, et diagnostiline modelleerimine peaks olema võimalik nii kõrgemal, näiteks arvuti käsusüsteemi tasandil kui ka madalamal, näiteks ventiil- või koguni transistorskeemide tasandil.

Digitaalsüsteemide diagnostikaprobleemid on võrdlemisi hästi läbitöötatud loogikaskeemide tasandil, Formaalseks matemaatiliseks aparaadiks sobib siin kõige paremini Boole'i diferentsiaalaparaat, mida traditsioonilistes *diskreetse matemaatika* või *arvutiteooria* kursustes harilikult ei käsitleta. Nimetatud aparaadist arusaamine aitab paremini mõista diagnostikaprobleemide kompleksi kui tervikut. Siiski pole sellest teooriast suurt kasu süsteemide uurimiseks kõrgematel tasanditel, kus bittide ja loogikatehete asemel kasutatakse niisugusi mõisteid nagu sõnad, mikroprogrammid, masinakäsud. Sobivaks modelleerimise vahendiks süsteemide diagnostilisel analüüsil kõrgematel tasanditel oleksid otsustusdiagrammid. Viimaste erijuhuks on binaarsed otsustusdiagrammid, mis võimaldavad diagnostilist analüüsi loogikatasemel.

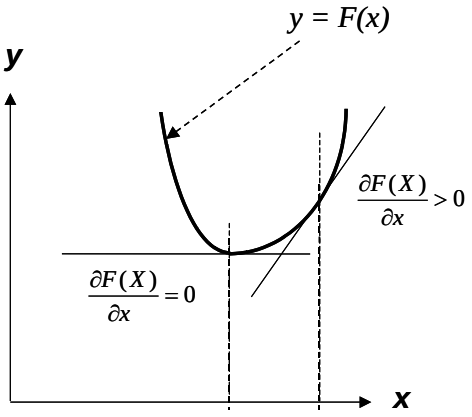
Käesolevas peatükis käsitlemegi nimetatud kahte modelleerimisviisi – Boole'i differentsiaalaparaati ja otsustusdiagramme.

1.1. Boole'i differentsiaalaparaat

Boole'i differentsiaalaparaati käsitletakse suhteliselt harva traditsioonilistes diskreetse matemaatika kursustes. Põhjalikumalt on neid vaadeldud töödes [Ake 59, Tha 71, Boc 75, Boc 89]. Peamiseks Boole'i differentsiaalarvutuse rakendusalaks on olnud digitaalskeemide uurimine dünaamikas. Käesolevas õppevahendis kasutame Boole'i differentsiaal-aparaati digitaalskeemide diagnostiliseks modelleerimiseks.

1.1.1. Boole'i osatuletised

Matemaatilises analüüsis mõistetakse funktsiooni $y = F(x)$ *tuletise* dy/dx all selle funktsiooni juurdekasvu ja argumendi juurdekasvu suhte piirväärtust argumendi juurdekasvu lähenemisel nullile. Geomeetrilise interpretatsiooni järgi on tuletis dy/dx funktsiooni graafiku dy/dx puutuja tõus punktis, mille abstsiss on x. Tuletist võib interpreteerida ka funktsiooni muutumise kiirusena. Joonisel 1-1 esitatud funktsiooni näitel on selge, et funktsiooni tuletis võib põhimõtteliselt omada igasuguseid väärtusi alates nullist kuni lõpmatuseni.



Joonis 1-1. *Funktsiooni tuletise geomeetriline interpretatsioon*

Analoogilise tuletise mõiste võib sisse viia ka Boole'i funktsioonide jaoks. Kuna aga *Boole'i funktsiooni* $y = F(x)$ puhul nii y kui ka x võivad omada vaid kahte väärtust 0 ja 1, siis ei saa me enam rääkida funktsiooni muutumise kiirusest, vaid võime rääkida vaid sellest, kas funktsiooni väärtus muutub argumendi väärtuse muutudes või ei muutu. Seega saab ka *Boole'i funktsiooni tuletise* väärtus omada vaid kahte väärtust 0 ja 1, kusjuures 0 vastaks sellele, et y-i väärtus ei muutu x-i väärtuse muutudes ja vastupidi, 1 vastaks sellele, et y-i väärtus muutub, kui x-i väärtus muutub.

Kuna Boole'i funktsioonid on tüüpiliselt n-muutuja funktsioonid $y = F(X)$, kus $X = (x_1,x_2,..., x_n)$, ja $n \geq 1$, siis tuleb siin kasutada *Boole'i osatuletise* mõistet, mis kirjeldab y-i muutumist mingi konkreetse $x_i \in X$ muutumise suhtes. Niisugust muutumist kirjeldab funktsioon

$$\frac{\partial F(X)}{\partial x_i} = F(x_1,...x_i,...x_n) \oplus F(x_1,...\bar{x}_i,...x_n) \qquad (1\text{-}1)$$

ehk lihtsamal kujul

$$\frac{\partial F(X)}{\partial x_i} = F(x_1,...x_i = 0,...x_n) \oplus F(x_1,...x_i = 1,...x_n) \qquad (1\text{-}2)$$

Boole'i funktsioon (1-2) leiab suurepärase rakenduse digitaalskeemide diagnostikas. Esitagu meil funktsioon $y = F(X) = F(x_1,x_2,..., x_n)$ kombinatsioonskeemi, millel on n sisendit. Olgu sellel skeemil sisend x_i katki, nii et signaal sealt kunagi läbi ei lähe. Matemaatiliselt võiks see tähendada näiteks, et $x_i \equiv 0$[1]. Kui nüüd rakendada selles sisendis signaali $x_i = 1$, siis rikke tõttu leiaks seal aset signaali x_k väärtuse muutus $1 \rightarrow 0$. Niisuguse rikke avastamiseks skeemi väljundis y huvitab meid olukord, kus ka y muutuks x_i muutudes. Lähtudes eelpool toodud Boole'i osatuletise interpretatsioonist saaksime taolist olukorda kirjeldada võrrandi

$$\frac{\partial F(X)}{\partial x_i} = 1 \qquad (1\text{-}3)$$

lahendite abil. Iga differentsiaalvõrrandi (1-3) lahend oleks testiks (ehk testvektoriks), mis avastab rikke $x_i \equiv 0$ funktsiooniga $y = F(X)$ esitatud kombinatsioonskeemi sisendis. Võrrandi lahendamiseks tuleks ta vasakpool

[1] Sõltuvalt tehnoloogiast võib see mõnikord tähendada ka teist olukorda, nimelt et $x_i \equiv 1$. Konkreetselt käsitleme seda probleemi edaspidi (vt. 2.1.4).

teisendada *disjunktiivseks normaalkujuks.* Iga konjuktsioon annaks siis meile ühe võimaliku lahendi.

Näide 1-1.

Vaatleme kombinatsioonskeemi joonisel 1-2.

Skeemi funktsiooniks on $y = x_1x_2 \vee x_3x_4$. Valemit (1-2) kasutades leiaksime võrrandi

$$\frac{\partial y}{\partial x_1} = x_3x_4 \oplus (x_2 \vee x_3x_4) = 1 \qquad (1\text{-}4)$$

Võrrandit (1-4) disjunktiivsele normaalkujule (DNK) teisendades saaksime

$$\frac{\partial y}{\partial x_1} = x_2\overline{x_3} \vee x_2\overline{x_4} = 1 \qquad (1\text{-}5)$$

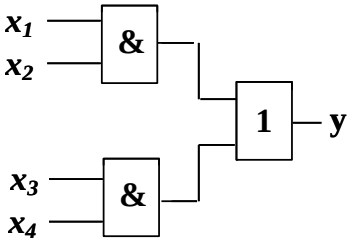
mis annaks kaks võimalikku test vektorit sisendi x_1 testimiseks:

$$x_1\,x_2\,x_3\,x_4 \;=\; \text{-10-}$$

ja

$$x_1\,x_2\,x_3\,x_4 \;=\; \text{-1-0,}$$

kus "-" tähendab, et väärtus jääb määramatuks (ehk pole oluline).



Joonis 1-2. *Kombinatsioonskeem*

Huvitav on siin märkida, et mõlema lahendi puhul jääb testitav muutuja x_1 ise määramatuks, mis peabki nii olema, sest kuna just muutujat

x_1 testime, siis peamegi talle põhimõtteliselt lubama mõlemat võimalikku väärtust: õiget ja väära (rikke korral).

1.1.2. Boole'i osatuletiste omadusi

Näitest 1-1 ilmneb, et võrrandi (1-4) teisendamine DNK-ks pole sugugi lihtne. Teisenduse lihtsustamiseks vaatame järgnevalt mõningaid Boole'i osatuletiste omadusi.

1. Olgu funktsioon $P(X)$ esitatav kujuna $P(X) = F(X) \wedge G(X)$, kusjuures osaavaldis $F(X)$ on sõltumatu muutujast $x_i \in X$. Siis on differentseeritavat avaldist võimalik lihtsustada järgneva teisenduse abil:

$$\frac{\partial[F(X)G(X)]}{\partial x_i} = F(X)\frac{\partial G(X)}{\partial x_i} \qquad (1\text{-}6)$$

2. Olgu funktsioon $P(X)$ esitatav kujuna $P(X) = F(X) \vee G(X)$, kusjuures $F(X)$ on sõltumatu muutujast $x_i \in X$. Siis on differentseeritavat avaldist võimalik lihtsustada järgnevalt:

$$\frac{\partial[F(X)+G(X)]}{\partial x_i} = \overline{F(X)}\frac{\partial G(X)}{\partial x_i} \qquad (1\text{-}7)$$

Üsna endastmõistetavad on ka järgmised kaks omadust:

$$\frac{\overline{\partial F(X)}}{\partial x_i} = \frac{\partial F(X)}{\partial x_i} \qquad (1\text{-}8)$$

$$\frac{\partial F(X)}{\partial \overline{x_i}} = \frac{\partial F(X)}{\partial x_i} \qquad (1\text{-}9)$$

Kasutades ülalesitatud omadusi, on võimalik üsna märgatavalt lihtsustada ka päris keerukate Boole'i avaldiste differentseerimist.

Näide 1-2.

Olgu vaja differentseerida muutuja x_5 järgi järgmist avaldist:

$$y = x_1x_2 \vee x_3(\overline{x_2}x_4 \vee \overline{x_1}(x_4 \vee (x_5 \vee \overline{x_2}x_6)) \vee x_1\overline{x_3} \qquad (1\text{-}10)$$

Kuna osaavaldis

$$F(X) = x_1 x_2 \vee x_1 \overline{x_3}$$

ei sõltu muutujast x_5, siis võime selle differentsiaali märgi alt välja viia, lihtsustades differentseeritavat osa. Kasutades vaheldumisi teisendusi (1-6) ja (1-7) saaksime:

$$\frac{\partial y}{\partial x_5} = \overline{\overline{x_1 x_2 \vee x_1 x_3}} \frac{\partial(x_3(\overline{x_2}x_4 \vee \overline{x_1}(x_4 \vee (x_5 \vee \overline{x_2}x_6))))}{\partial x_5} =$$
$$= \overline{\overline{x_1 x_2 \vee x_1 x_3} x_3} \frac{\partial(\overline{x_2}x_4 \vee \overline{x_1}(x_4 \vee (x_5 \vee \overline{x_2}x_6)))}{\partial x_5} =$$
$$= \overline{\overline{x_1 x_2 \vee x_1 x_3} x_3 \overline{x_2 x_4}} \frac{\partial(\overline{x_1}(x_4 \vee (x_5 \vee \overline{x_2}x_6)))}{\partial x_5} =$$
$$= \overline{\overline{x_1 x_2 \vee x_1 x_3} x_3 \overline{x_2 x_4} x_1} \frac{\partial(x_4 \vee (x_5 \vee \overline{x_2}x_6))}{\partial x_5}$$
$$= \overline{\overline{x_1 x_2 \vee x_1 x_3} x_3 \overline{x_2 x_4} x_1 x_4 \overline{x_2 x_6}} \frac{\partial x_5}{\partial x_5}$$

Kuna $\partial x_5/\partial x_5 \equiv 1$, siis olemegi lahti saanud tülikast operatsioonist $\oplus$ ja testide leidmine avaldisele (1-10) muutuja x_5 testimiseks taandub nüüd juba DeMorgani teisendustele ning sulgude avamisele:

$$\frac{\partial y}{\partial x_5} = \overline{(x_1 x_2 \vee x_1 \overline{x_3})} x_3 \overline{(\overline{x_2 x_4})}\, \overline{x_1}\, \overline{x_4}\, \overline{(\overline{x_2 x_6})} \frac{\partial x_5}{\partial x_5} =$$
$$= (\overline{x_1} \vee \overline{x_2})(\overline{x_1} \vee x_3) x_3 (x_2 \vee \overline{x_4}) \overline{x_1}\, \overline{x_4} (x_2 \vee \overline{x_6}) =$$
$$= \overline{x_1}\, \overline{x_4} x_3 x_2 \vee \ldots = 1$$

Üheks võimalikuks testvektoriks leiame $x_1\ x_2\ x_3\ x_4\ x_5$ = 0110--. Olgu märgitud, et testide genereerimisel praktilisest aspektist lähtudes, ei olegi vaja tervet disjunktiivset normaalkuju kätte saada kõikide lahendite leidmiseks, vaid piisab ju ka ühest ainsast testvektorist.

1.1.3. Boole'i vektortuletised

Mitmetel juhtudel on vaja analüüsida Boole'i funktsiooni tundlikkust mitme muutja väärtuse üheaegse muutumise suhtes. Niisugusteks juhtudeks on näiteks:

- mitme rikke üheaegne olemasolu skeemis või
- ühest ja samast rikkest tuleneva vigase signaali levimine mitme hargneva signaalitee kaudu skeemis.

Testvektorite leidmiseks niisugusel juhul võib kasutada *vektortuletisi*:

$$\frac{\partial F(X)}{\partial(x_i, x_j)} = F(x_1,...x_i, x_j,...x_n) \oplus F(x_1,...\bar{x}_i, \bar{x}_j,...x_n) \quad (1\text{-}11)$$

Näide 1-3.

Vaatleme taas skeemi joonisel 1-2, mille funktsiooniks on $y = x_1x_2 \vee x_3x_4$. Olgu korraga vigased nii sisend x_2 kui ka x_3. niisuguse 2 rikke põhjuseks võib olla ka üksainus rike skeemis, millest tingitud veasignaal levib mööda kahte haru. Testvektori niisuguse kordse vea avastamiseks leiame võrrandist:

$$\frac{\partial F(X)}{\partial(x_2, x_3)} = x_1\bar{x}_4 \vee \bar{x}_1x_4 \vee x_1x_4(\bar{x}_2\bar{x}_3 \vee x_2x_3) = 1$$

Võrrandil on neli lahendit:

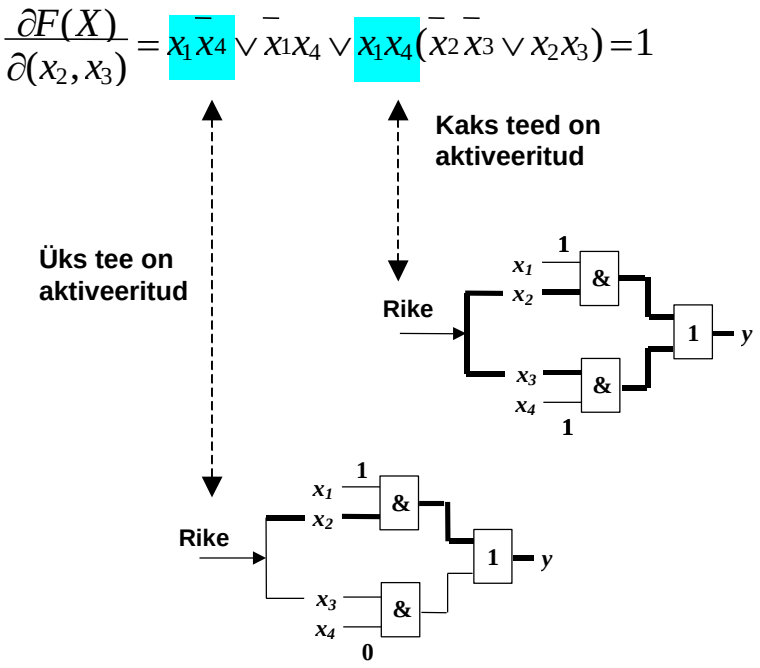
1) 1--0,
2) 0--1,
3) 1001,
4) 1111.

Lahenditel on ka oma kindel "füüsikaline" tähendus (vt. ka joonist 1-3).

Esimene lahend 1--0 lubab levida ühel ainsal veasignaalil x_2 läbi skeemi, blokeerides teise veasignaali x_3 levi. Sama omadus on ka teisel lahendil, kusjuures nüüd levib viga vaid sisendist x_3. Mõlemal juhul võivad signaalimuutused sisendites x_2 ja x_3 olla suvalised $0 \to 1$ või $1 \to 0$ (seega kokku 4 võimalust).

Kolmanda ja neljanda lahendi puhul on aga aktiveeritud mõlemad harud vea levikuks, kusjuures mitte enam kõik vigade kombinatsioonid pole enam avastatavad: väljundisse levib viga vaid siis, kui mõlemad veasignaalid on

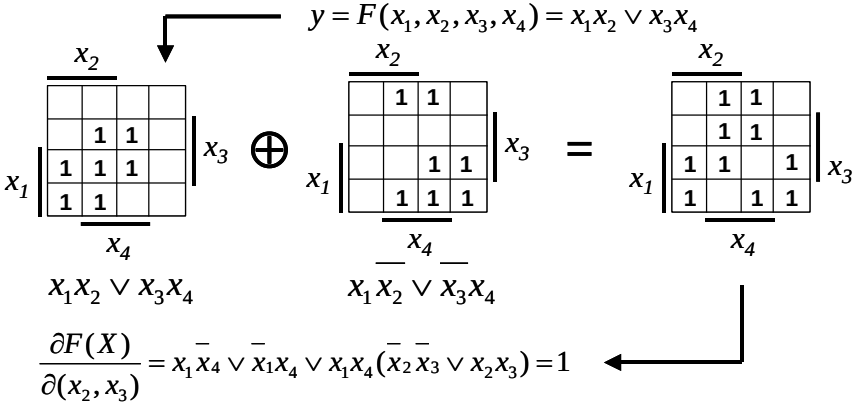
ühesuunalised. Kui veasignaalid muutuvad vastupidises suunas, siis nad kompenseerivad teineteist ja vead jäävad avastamata (vead *maskeeruvad*).



Joonis 1-3. *Vektortuletise füüsikaline interpretatsioon*

Näites 1-3 vaadeldud olukord illustreerib ühte keerukamat probleemi testide genereerimisel. Kui *veasignaal* levib mitut haru pidi, oleks vaja tegeleda maskeerumise kontrolliga, mis teeb testide genereerimise algoritmi keerukamaks ja aeglasemaks. Algoritm lihtsustuks, kui lubaksime veasignaale levida alati vaid ühte haru pidi, blokeerides teised harud. Samas aga, kui niiviisi teeksime, sünnib oht välistada testi leidmist juhul, kui üksnes mitme haru kaudu veasignaale levitav test eksisteerib.

Nn. lihtsustatud testide genereerimise algoritmid, mis aktiveerivad vaid üksikuid teid, korreleeruvad hästi ka faktiga, et Boole'i vektortuletisi arvutada on väga keerukas. Üheks võimaluseks neid arvutada on *Carnaugh' kaartide* kasutamine (joonis 1-4).



Joonis 1-4. Vektortuletiste arvutamine Carnaugh' kaardi abil

1.1.4. Kordsed Boole'i osatuletised

Olgu antud Boole'i funktsioon $y = F(X) = F(x_1,x_2,..., x_n)$. Kordne tuletis kahe muutuja x_i ja x_j järgi analoogiliselt traditsioonilise matemaatilise analüüsiga on defineeritud järgmiselt:

$$\frac{\partial^2 y}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(\frac{\partial y}{\partial x_j}\right) \qquad (1\text{-}12)$$

Kordset Boole'i osatuletist saab kasutada *kordsete rikete* analüüsiks, näiteks testide genereerimiseks kordsele rikkele. Olgu skeemis, millele vastab funktsioon $y = F(x_1,x_2,..., x_n)$, vigased kaks sisendit: x_i ja x_j. Võiks püstitada ülesande genereerida test, mis avastaks ühe kahest rikkest, näiteks rikke sisendis x_j tingimusel, et teine rike sisendis x_i ei maskeeriks esimest.

Rikke avastamise tingimuseks sisendis x_j oleks

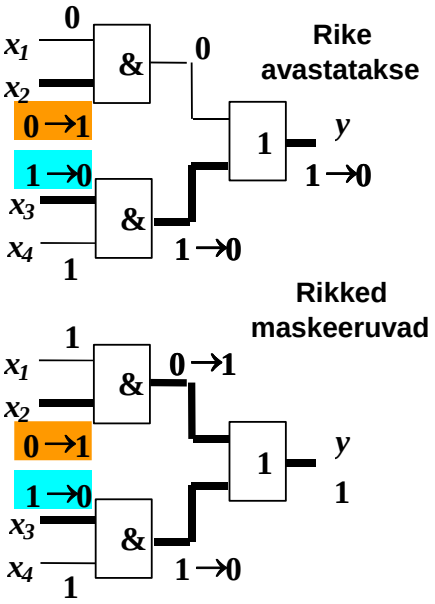
$$\frac{\partial y}{\partial x_j} = F_1(X) = 1 \qquad (1\text{-}13)$$

Selleks et viga sisendis x_i ei saaks mõjutada tingimust (1-13) s.t. maskeerida riket sisendis x_j, tuleks lahendada võrrand

$$\frac{\partial F_1(X)}{\partial x_i} = 0 \qquad (1\text{-}14)$$

Ehk teiste sõnadega, tuleks lahendada võrrandisüsteem:

$$\frac{\partial^2 y}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(\frac{\partial y}{\partial x_j}\right) = 0$$
$$\frac{\partial y}{\partial x_j} = 1 \qquad (1\text{-}15)$$



Joonis 1-5. *Kordse tuletise kasutamine rikete maskeerumise vältimiseks*

Näide 1-4.

Olgu antud kombinatsioonskeem, millele vastab funktsioon $y = x_1x_2 \vee x_3x_4$ ja kus on korraga vigased nii sisend x_2 kui ka x_3. Joonisel 1-5 on illustreeritud kaks juhtu. Ülemisel skeemil illustreeritakse testi $x_1x_2x_3x_4 = 0011$, mille saame saame lahendades võrrandisüsteemi (1-15):

$$\frac{\partial^2 y}{\partial x_2 \partial x_3} = \frac{\partial}{\partial x_2}\left(\frac{\partial y}{\partial x_3}\right) = x_1 x_4 = 0$$

$$\frac{\partial y}{\partial x_3} = \overline{x_1} x_4 \vee \overline{x_2} x_4 = 1$$

Alumisel skeemil aga illustreeritakse olukorda, kus on küll genereeritud test $x_1x_2x_3x_4 = 1011$ rikkele sisendis x_3 võrrandi (1-13) lahendamisega:

$$\frac{\partial y}{\partial x_3} = \overline{x_1} x_4 \vee \overline{x_2} x_4 = 1 \quad \longrightarrow \quad \overline{x_2} x_4 = 1$$

aga rike jääb maskeerituks teise rikke tõttu sisendis x_2, kuna pole täidetud tingimus (1-14).

1.1.5. Liitfunktsioonide diferentseerimine

Keerukamate digitaalskeemide puhul ei ole otstarbekas neid esitada väljundite Boole'i funktsioonide süsteemina kahel põhjusel:

- väljundfunktsioonide keerukuse tõttu muutuvad ka Boole'i differentsiaal-võrrandid liiga keeruliseks;
- diagnostika aspektist ei piisa ainult sisendite testimisest, vaid on vaja testida ka skeemi sisepunkte.

Seetõttu esitatakse digitaalskeeme enamasti komponentide (või alamskeemide) võrguna, kus igale komponendile (või alamskeemile) vastab mingi lihtsam Boole'i funktsioon. Välundfunktsioone saab sellisel juhul vaadelda liitfunktsioonidena, mille argumentideks võivad olla nii sisendmuutujad kui ka skeemi sisepunktidele vastavad funktsioonid.

Vaatleme skeemi joonisel 1-6, mis on esitatud *liitfunktsioonina*

$$y = F(X) = F(X,x_k) = F(X, F_k(X_k)),$$

kus

$$x_k = F_k(X_k)$$

$$X_k \subseteq X.$$

Niisuguse funktsiooni tuletis mingi muutuja $x_i \in X_k$ järgi võib avaldada Boole'i funktsioonide korrutisena:

$$\frac{\partial y}{\partial x_i} = \frac{\partial F(X, F_k(X_k))}{\partial x_i} = \frac{\partial F(X, x_k)}{\partial x_i} = \frac{\partial y}{\partial x_k}\frac{\partial F_k(X_k)}{\partial x_i} = \frac{\partial y}{\partial x_k}\frac{\partial x_k}{\partial x_i} \quad (1\text{-}16)$$

Üldjuhul, kui liitfunktsioon on esitatud *superpositsioonide* ahelana

$$y = F_k(F_{k-1}(\ldots F_2(x_1,X)\ldots,X), \quad (1\text{-}17)$$

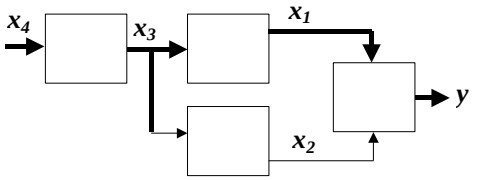
saame korrutise

$$\frac{\partial y}{\partial x_1} = \frac{\partial F_k}{\partial F_{k-1}}\frac{\partial F_{k-1}}{\partial F_{k-2}}\cdots\frac{\partial F_2}{\partial x_1} \quad (1\text{-}18)$$

Olgu märgitud aga, et niisugune *ahelreegel* liitfunktsiooni osatuletise esitamiseks on lihtsustatud ja kehtib ainult erijuhul, mis diagnostika aspektist vastab sellele, et veasignaal levib vaid ühte teed pidi.

Näide 1-5.

Vaatleme skeemi joonisel 1-6, kus eraldi on välja toodud 4 olulist muutujat, teisi muutujaid hulgast X pole näidatud.



Joonis 1-6. *Testide genereerimine liitfunktsiooni osatuletiste abil*

Arvutades tuletist $\partial y/\partial x_4$ võime kasutada liitfunktsiooni tuletise arvutamise ahelreeglit

$$\frac{\partial y}{\partial x_4} = \frac{\partial y}{\partial x_1}\frac{\partial x_1}{\partial x_3}\frac{\partial x_3}{\partial x_4}$$

aga ainult sel juhul, kui veasignaali levik paralleelselt teisi harusid kaudu on blokeeritud. Selleks võib kasutada kahte võimalikku lisatingimust:

1) kordsete tuletiste põhimõtet

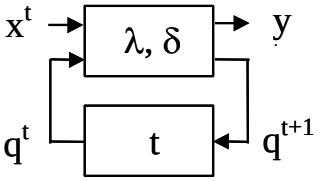
$$\frac{\partial y}{\partial x_2}\left(\frac{\partial y}{\partial x_1}\right) = 0$$

2) või veasignaali leviku blokeerimist juba enne komponenti $y = F(x_1,x_2,X)$

$$\frac{\partial x_2}{\partial x_3} = 0$$

1.1.6. Boole'i osatuletised järjestikskeemidele

Järjestikskeemide esitamiseks kasutatakse *Huffmann'i mudelit* (joonis 1-7), millele vastab kaks tüüpi funktsioone: *väljundfunktsioon* ja *siirdefunktsioon*.



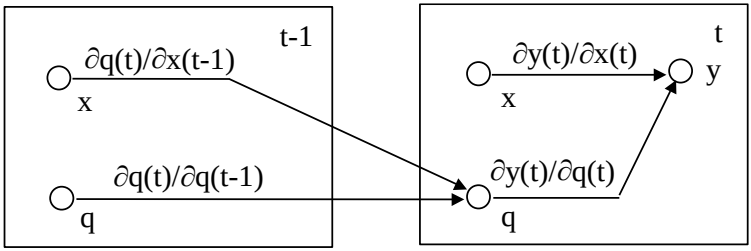
$$y(t) = \lambda[x(t),q(t)]$$

$$q(t+1) = \delta[x(t),q(t)]$$

Joonis 1-7. *Järjestikskeemide mudel*

Mõlemad funktsioonid on tavalised Boole'i funktsioonid, kus aeg t on parameetriks ja seob konkreetseid muutujate väärtusi erinevate taktidega. Nii tulebki vaadelda ühte muutujat $x(t)$ parameetri t eri väärtuste korral t_1 ja t_2 sisuliselt kui kahte erinevat muutujat $x(t_1)$ ja $x(t_2)$. Öeldut arvesse võttes toimub niisuguste *ajast sõltuvate Boole'i funktsioonide* diferentseerimine

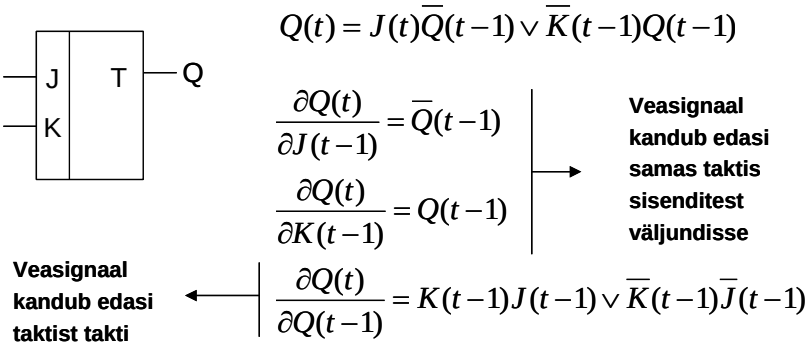
analoogiliselt ajast sõltumatute Boole'i funktsioonidega. Joonisel 1-8 on illustreeritud Huffmann'i mudelist tulenevad neli Boole'i osatuletise tüüpi.



Joonis 1-8. *Järjestikskeemide osatuletiste tüübid*

Näide 1-6.

Joonisel 1-9 on esitatud *JK-trigeri* funktsioon ja sellest tulenevad Boole'i osatuletised.

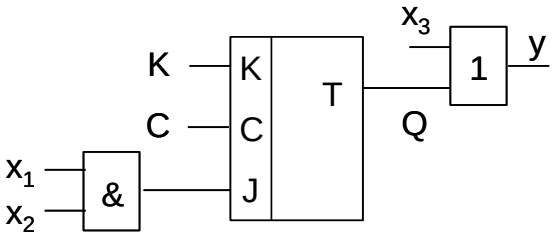


Joonis 1-9. *JK-triggeri Boole'i osatuletised*

Kui seni vaatlesime Boole'i differentsiaalvõrrandite kasutamist üksikute testvektorite genereerimiseks, siis ajafunktsioonide differentseerimise eripäraks on võimalus formaliseerida järjestikskeemide *testjadade* genereerimist.

Näide 1-7.

Genereerime testi joonisel 1-10 esitatud järjestikskeemi sisendile x_2 kasutades Boole'i tuletiste arvutamise ahelmeetodit võrrandi (1-19) kujul.



Joonis 1-10. *Järjestikskeem*

$$\frac{\partial y(t)}{\partial x_1(t-\tau)} = \frac{\partial y(t)}{\partial Q(t)} \frac{\partial Q(t)}{\partial J(t-1)} \frac{\partial J(t-1)}{\partial x_1(t-1)} = \overline{x_3}(t)\overline{Q(t-1)}x_2(t-1) = 1 \quad (1\text{-}19)$$

Võrrandi vasakut poolt koostama hakates polnud veel teada testjada pikkus, seetõttu sai ajaline nihe muutujate y ja x_1 vahel fikseeritud esialgu veel määramatu väärtusega sümbolmuutujaga τ. Liitfunktsiooni osatuletiste jada valmimisel selgub ka selle muutuja lõplik väärtus ehk otsitav ajaline nihe: $\tau = 1$.

Võrrandi vasakul pool olevast tingimusest $Q(t\text{-}1) = 0$ tuleneb uus tingimus $K(t\text{-}2) = 1$. Asendades nüüd selle tingimuse võrrandi (1-19) vasakusse poolde, saame lõpliku võrrandi,

$$x_3(t)K(t-2)x_2(t-1) = 1 \quad (1\text{-}20)$$

millest tuleneb Tabelis 1-1 esitatud 3-taktiline testjada.

Tabel 1-1. *3-taktiline test jada skeemile joonisel 1-10*

t	x_1	x_2	x_3	Q'	K	y
1					1	
2	D	1		0		
3			0	D		D

Tühjad ruudud tabelis vastavad määramata väärtustele. *Sümboliga D* tähistame rikke tõttu potentsiaalselt muutuvat väärtust. Apostroof muutuja *Q* juures tähendab muutuja väärtust eelmises taktis.

1.1.7. Boole'i differentsiaalid

Testeksperimentide läbiviimist ja rikete lokaliseerimist (diagnoosimist) saab modelleerida *Boole'i differentsiaalide* abil.

Klassikalisest matemaatilisest analüüsist teame, et funktsiooni $y = F(x)$ *differentsiaaliks*

$$dy = \frac{\partial y}{\partial x} dx \tag{1-21}$$

nimetatakse selle funktsiooni juurdekasvuga ekvivalentset lõpmatult vähenevat suurust, mis on võrdeline argumendi juurdekasvuga, kusjuures funktsioonil on differentsiaal olemas siis ja ainult siis, kui funktsioon on differentseeruv. Boole'i funktsioonide puhul, kus argumentide arv on suur, kasutame taas *osadifferentsiaali* mõistet:

$$d_x y = \frac{\partial y}{\partial x} dx$$

Boole'i differentsiaali dx all mõistame muutuja x väärtuse muutumist mingi rikke tõttu: $dx = 1$, kui x väärtus muutub (esines viga) ja $dx = 0$, kui x väärtus ei muutu (viga ei olnud). Boole'i funktsiooni osadifferentsiaal $d_x y$ väljendab testeksperimendi tulemust: $d_x y = 1$, kui y väärtus muutub x väärtuse muutudes ja $d_x y = 0$, kui ei muutu. Tingimuseks, et niisugune muutus toimub ongi eelpool käsitletud osatuletise võrdumine 1-ga: $\partial y/\partial x=1$.

Boole'i funktsiooni *täisdifferentsiaal* avaldub valemina:

$$\begin{gathered} dy = F(X) \oplus F(X \oplus dX) = \\ = F(x_1,..., x_i,...x_n) \oplus F(x_1 \oplus dx_1,..., x_i \oplus dx_i,...x_n \oplus dx_n) \end{gathered} \tag{1-22}$$

kus

$$dX = (dx_1, dx_2,..., dx_n).$$

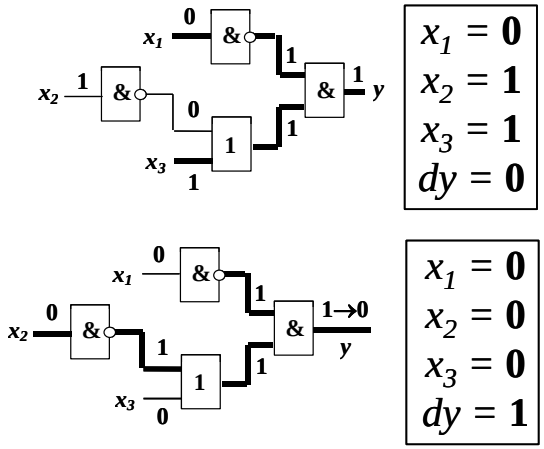
Võrreldes valemit (1-22) valemiga (1-11), on kerge näha, et Boole'i täisdifferentsiaal kujutab endast kõigi Boole'i vektortuletiste hulga parameetrilist kuju. Täpsemalt väljendub nimetatud omadus järgmises täisdifferentsiaali kujus

$$dy = \frac{\partial y}{\partial x_1} dx_1 \overline{dx_2} \ldots \overline{dx_n} \vee \ldots \vee \frac{\partial y}{\partial x_n} \overline{dx_1}\,\overline{dx_2} \ldots \overline{dx_{n-1}} dx_n \vee \ldots \vee \frac{\partial y}{\partial (x_1 x_2)} dx_1 dx_2 \overline{dx_3} \ldots \overline{dx_n} \vee \ldots \vee \frac{\partial y}{\partial (x_1 x_2 \ldots x_n)} dx_1 dx_2 \ldots dx_n \qquad (1\text{-}23)$$

või näiteks kahe muutuja funktsiooni $y = F(x_1,x_2)$ puhul

$$dy = \frac{\partial y}{\partial x_1} dx_1 \overline{dx_2} \vee \frac{\partial y}{\partial x_2} \overline{dx_1} dx_2 \vee \frac{\partial y}{\partial (x_1 x_2)} dx_1 dx_2 \qquad (1\text{-}24)$$

Põhiline Boole'i täisdifferentsiaali roll digitaalskeemide diagnostilises analüüsis seisneb võimaluse loomises dešifreerida *diagnostikaeksperimentide* tulemusi, teiste sõnadega - määrata vigaste väljundsignaalide põhjusi.



Joonis 1-11. *Kaks diagnostikaeksperimenti*

Näide 1-8.

Olgu antud kombinatsioonskeem funktsiooniga

$$y = \bar{x}_1(\bar{x}_2 \vee x_3) \tag{1-25}$$

ja selle täisdifferentsiaaliga

$$dy = y \oplus (\bar{x}_1 \oplus dx_1)((\bar{x}_2 \oplus dx_2) \vee (x_3 \oplus dx_3)) \tag{1-26}$$

Viime läbi kaks diagnostikaeksperimenti (vt. joonis 1-11) testvektoritega T_1 = 011 ja T_2 = 000. Oletame, et esimesel juhul test "läheb läbi", s.t. väljundi väärtus on y = 1, mis vastab oodatavale tulemusele. Eksperimendi tulemuseks saame seega dy = 0. Oletame, et teisel juhul test "ei lähe läbi", s.t. väljundis avastatakse oodatava signaali 1 asemel 0. *Diagnostikaksperimendi tulemus* vastab nüüd differentsiaalile dy = 1.

Asetame vektoritega T_1 ja T_2 määratud sisendmuutujate väärtused ning testide tulemused dy avaldisse (1-26) ning saame kahe testeksperimendiga määratud võrrandisüsteemi:

$$\begin{cases} dy = 1 \oplus \overline{dx}_1(dx_2 \vee \overline{dx}_3) = 0 \\ dy = 1 \oplus \overline{dx}_1(\overline{dx}_2 \vee dx_3) = 1 \end{cases}$$

Lihtsustades avaldisi ja teisendades ka esimese avaldise tõeseks, saame

$$\begin{cases} \overline{dx}_1^{\,0}(dx_2^1 \vee \overline{dx}_3^{\,1}) = 1 \\ dx_1^0 \vee dx_2^0 \overline{dx}_3^{\,0} = 1 \end{cases} \tag{1-27}$$

Et differentsiaale muutujate erinevate väärtuste tõttu üheselt mõista viime diferentsiaalide tähistusse sisse ülaindeksid, mis märgivad muutujate õigeid väärtusi. Näiteks $dx^1{}_2$ = 1 tähendab, et muutuja x_2 väärtus peaks olema 1, aga rikke tõttu on see tegelikult 0.

Süsteemi (1-27) esimesest võrrandist tuleneb üheselt mõistetav tulemus:

$$\overline{\partial x}_1^{\,0} = 1$$

mis tähendab, et sisendis x_1 kindlasti puudub rike $x_1 \equiv 1$. Rohkem üheselt mõistetavaid fakte ei tulene enam kummastki süsteemi (1-27) võrrandist. Küll aga tuleb rohkesti informatsiooni, kui kaks võrrandit loogiliselt omavahel korrutada, mis võtab kokku mõlemast eksperimendist saadava informatsiooni:

$$\overline{dx}_1^0 (dx_2^1 dx_2^0 \overline{dx}_3^0 \vee dx_2^0 \overline{dx}_3^0 \overline{dx}_3^1) = 1 \qquad (1\text{-}28)$$

Kasutades lihtsustust

$$dx_k^0 dx_k^1 = 0 \qquad (1\text{-}29)$$

mis tähendab, et ühes ja samas sisendis (või ühel ja samal juhtmel) x_k ei saa olla korraga mõlemat tüüpi rikked $x_k \equiv 1$ ja $x_k \equiv 0$, saame võrrandist (1-28) järgmise tulemuse:

$$\overline{dx}_1^0 dx_2^0 \overline{dx}_3^0 \overline{dx}_3^1 = 1 \qquad (1\text{-}30)$$

Tõene lause (1-30) kujutabki endast *diagnoosi* ehk kahe testeksperimendi tulemuse lahtimõtestamist. Lause (1-30) väidab, et

- sisendis x_1 puudub rike $x_1 \equiv 1$,
- sisendis x_2 on rike $x_2 \equiv 1$, ja
- sisend x_3 on korras (mõlemat tüüpi rikked puuduvad).

Boole'i täisdiferentsiaali omadused:

Olulisteks *differentsiaalioperaatori d* omadusteks on:

- Boole'i võrrandite differentseerimise võimalus, ja
- Boole'i täisdifferentsiaali avaldise *invariantsus*.

Olgu antud Boole'i võrrand

$$F_1(X) = F_1(X).$$

Siis kehtib ka järgmine võrrand

$$dF_1 = dF_2.$$

Niisugust tehet nimetatakse Boole'i võrrandi täisdifferentseerimiseks.

Olgu antud liitfunktsioon

$$F_k(X, F_i(X)).$$

Siis, vastavalt Boole'i täisdiferentsiaali definitsioonile (1-22), saame

$$dF_k(X, F_i(X)) = F_k(X, F_i(X)) \oplus F_k(X \oplus dX, F_i(X \oplus dX))).$$

Aga kuna

$$F_i(X \oplus dX) = F_i(X) \oplus dF_i,$$

siis

$$dF_k = F_k(X, F_i(X)) \oplus F_k(X \oplus dX, F_i \oplus dF_i))). \qquad (1\text{-}31)$$

Avaldisest (1-31) järgneb, et üldine formaalne Boole'i täisdiferentsiaali avaldis ei sõltu sellest, kas muutujad on sõltumatud või ei ole sõltumatud. Teiste sõnadega, Boole'i täisdifferentsiaali avaldis on invariantne oma muutujate suhtes.

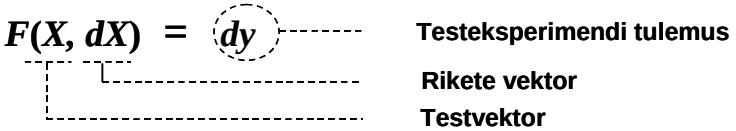
Sellel Boole'i täisdifferentsiaali invariantsuse omadusel põhineb võimalus arvutada differentsiaale ka liitfunktsioonidele. See aga omakorda laseb keerukaid Boole'i avaldisi tükeldada lihtsamateks osadeks ja sel moel lihtsustada differentseerimist.

1.1.8. Digitaalskeemide diagnostika üldvõrrand

Boole'i funktsioonide täisdifferentsiaali võrrand kätkeb endas võimalust lahendada kõiki digitaalskeemide diagnostikaga seotud ülesandeid. Kaheks põhiülesandeks loetakse:

- testide genereerimist (kui *pöördülesannet*) ja
- rikete diagnoosi (kui *otseülesannet*).

Kõigepealt kujutab täisdiferentsiaal endast testeksperimendi mudelit, kus on ühendatud korraga nii signaalide mõjutamine (stimuleerimine) testvektoritega, rikete modelleerimine ja testi tulemuste arvesse võtmine (joonis 1-12).



Joonis 1-12. *Diagnostika üldvõrrand*

Diagnostika üldvõrrandina esitatavast *diagnostika eksperimentide mudelist* tulenevad otseselt kahe diagnostika põhiülesande formuleeringud, mis seisnevad kahte tüüpi võrrandi lahendite leidmises (joonis 1-13):

Otseülesanne:
Testide genereerimine: *dX, dy = 1* **- antud**
X **- tuleb leida**
Pöördülesanne:
Rikete diagnoos: *X, dy* **- antud**
dX **– tuleb leida**

Joonis 1-13. *Diagnostika põhiülesannete tulenemine üldvõrrandist*

Rikete simuleerimist võib vaadelda kui rikete diagnoosi ülesande erijuhtu, kus imiteeritakse "tahtlikult" eksperimendi viga *dy* = 1 ja viiakse läbi sel põhjal diagnoos: leitakse millised vektoriga *dX* määratud rikked avastab antud test *X*.

Näide 1-9.

Vaatleme veelkord funktsiooni

$$y = \bar{x}_1(\bar{x}_2 \vee x_3)$$

täisdifferentsiaali

$$dy = y \oplus (\bar{x}_1 \oplus dx_1)((\bar{x}_2 \oplus dx_2) \vee (x_3 \oplus dx_3)) \quad (1\text{-}32)$$

Testide genereerimisel antakse ette mingi rikete kombinatsioon, näiteks *dX* = 011, mis tähendab, et soovitakse genereerida testi kordsele

rikkele sisendites x_2 ja x_3. Rikke avastamine tähendab ühtaegu veel ka tingimust $dy = 1$.

Võrrand (1-32) võtab nüüd kuju

$$y \oplus \overline{x_1}(x_2 \vee \overline{x_3}) = 1. \qquad (1\text{-}33)$$

Võrrandi (1-33) lahendiks on test, mis avastab ette antud rikete kombinatsiooni. Kerge on näha, et sama ülesande formuleeringu oleksime saanud ka kasutades vektorosatuletisi (vt. p.1.1.3).

Rikete diagnostika ülesande lahendamisel antakse ette mingi testvektor, näiteks $X = 011$, ja testi tulemus, näiteks $dy = 1$. Asetades nimetatud väärtused võrrandisse (1-32), saame uue lihtsustatud võrrandi.

$$y \oplus \overline{dx_1}(dx_2 \vee \overline{dx_3}) = 1. \qquad (1\text{-}34)$$

Rikete diagnoos leitakse võrrandi (1-34) lahendist.

Praktikas nii otse- kui ka pöördülesandeid enamasti lihtsustatakse. Testide genereerimisel antakse ette vaid üksik konkreetne rike $dx_k = 1$ ja kõik ülejäänud rikete avaldumiste võimalused välistatakse (kõigi ülejäänud muutujate $dx_i \in dX$ jaoks võetakse $dx_i = 0$). Sellisel juhul, nagu selgelt ilmneb valemist (1-23), redutseerub täisdifferentsiaali üldavaldis Boole'i osatuletiseks.

1.2. Otsustusdiagrammid

Boole'i differentsiaalaparaadi puuduseks on see, et ta on kasutatav üksnes digitaalsüsteemide abstraktse esituse loogikatasandil. Palju laiemaid võimalusi annab digitaalsüsteemide graafiline esitus *otsustusdiagrammide* (OD) kujul, kus kõik operatsioonid taanduvad graafi (otsustusdiagrammi *topoloogilisele analüüsile*). Otsustusdiagrammide erijuhuks on *binaarsed otsustusdiagrammid* (BOD).

Binaarsete otsustusdiagrammide töötlemiseks diagnostika aspektist on levinud kaks kontseptsiooni: otsustusdiagrammide manipuleerimine ehk loogikaoperatsioonide läbiviimine ja graafidel läbiviidav topoloogiline analüüs. Esimesel juhul ei välju me loogika algebra raamidest ja graafidega opereerimise meetodite üldistamine digitaalsüsteemide kõrgematele

tasanditele pole võimalik. Teisel juhul, kus graafidega töö baasiks on topoloogiline analüüs, on meetodite üldistamine loogikatasandilt kõrgematele tasanditele suhteliselt lihtne.

Käesolevas õppevahendis käsitleme otsustusdiagrammide analüüsi topoloogilist kontseptsiooni, et graafide abil mõista digitaalsüsteemide diagnostika ühtsust nii süsteemide madalatel kui ka kõrgematel esitustasanditel.

1.2.1. Binaarsete otsustusdiagrammide mõiste

1959. aastal võttis esmakordselt binaarsed otsustusdiagrammid kasutusele C.Y.Lee [Lee 59] nime all *binaarsed otsustusprogrammid*. 1976. aastal kasutati esmakordselt BOD mudelit digitaalskeemide diagnostikas Tallinna Tehnikaülikoolis [Uba 76], tookord *alternatiivsete graafide* nime all. Tegelikult veel varemgi, kuna juba 1974. aastal kaitses oma diplomitöö BOD kasutamisest digitaalskeemide simuleerimise kiirendamiseks TTÜ-s Virve Vaher käesoleva õppevahendi autori juhendamisel. Paar aastat hiljem võeti samad graafid kasutusele ka USAs *binaarsete otsustusdiagrammide* nime all [Ake 78]. Täna on BOD teooria väga laialt kasutatav ja väga kiiresti edasi arenev [Bry 86, Min 96, Sas 96, Dre 98].

Boole'i funktsiooni

$$y = F(X) = F(x_1,x_2,\ldots,x_n)$$

esitav BOD on *orienteeritud tsükliteta graaf*

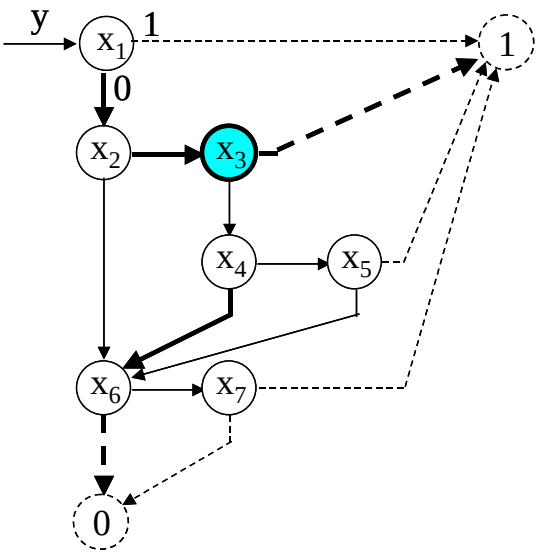
$$G_y = (M, X, \Gamma),$$

kus $M = \{m\}$ on *graafi tippude hulk*, $X = \{x_1,x_2,\ldots,x_n\}$ on funktsiooni argumentide hulk, $x(m) \in X$ on muutuja, mis on vastavusse seotud tipuga m ja $\Gamma(M,X)$ on funktsiooniga $y = F(X)$ üheselt määratud relatsioon, kus $\Gamma(m,x(m))$ määrab tipu m *järglase* vastavalt *tipumuutuja* $x(m)$ väärtusele. Vektoriga $X^* \in \{0,1\}^n$ on graafis määratud liikumine tipust tipu. Igal tipul m on kaks väljuvat *kaart* (*0-kaar* ja *1-kaar*) ning vastavalt kaks *naabrit* m^e, $e \in \{0,1\}$. *0-kaar* viib väärtusel $x(m) = 0$ tippu m^0 ($\Gamma(m,0) = m^0$) ja *1-kaar* väärtusel $x(m) = 1$ tippu m^1 ($\Gamma(m,1) = m^1$). Graafis on kaks *terminaaltippu* $m^{T,e}$, $e \in \{0,1\}$, mida me nimetame *0-terminaliks* $m^{T,0}$ ja *1-terminaliks* $m^{T,1}$.

Me ütleme, et tipumuutuja $x(m)$ väärtus aktiveerib tipu väljundkaare. Kui $x(m) = 1$, siis on aktiveeritud 1-kaar, ja kui $x(m) = 0$, siis on aktiveeritud 0-kaar. Mingi *tee on* graafis *aktiveeritud* siis, kui kõik seda teed moodustavad kaared on aktiveeritud. BOD on aktiveeritud väärtusele 1 (0),

siis kui graafis on aktiveeritud tee algtipust (juurest) 1-terminali (0-terminali).

Binaarne otsustusdiagramm $G_y = (M, X, \Gamma)$ esitab Boole'i funktsiooni $y = F(X)$, siis ja ainult siis, kui iga vektori $X^* \in \{0,1\}^n$ jaoks, mis viib graafis terminaaltippu $m^{T,e}$, kehtib $y = F(X^*) = e$.



Joonis 1-14. *Binaarne otsustusdiagramm*

Näide 1-10.

Joonisel 1-14 on esitatud BOD, mis vastab Boole'i funktsioonile

$$y = x_1 \vee x_2(x_3 \vee x_4 x_5) \vee x_6 x_7$$

Edaspidi esitame graafe selliselt, et 1-kaared lähevad paremale ja 0-kaared alla. Sellisel juhul ei ole vaja kaartele panna tähistusi 0 ja 1. Olgu antud vektor $X = 0110101$. Liikudes selle vektori järgi läbime tippe x_1, x_2, x_3,1. Terminaaltippu 1 jõudmine tähendab, et Boole'i funktsiooni väärtus antud vektori korral on 1.

Binaarseid otsustusdiagramme võib interpreteerida järgmistel viisidel:

- BOD on *binaarne programm* Boole'i funktsioonide arvutamiseks, kus tippudele vastavad *if-then-else* käsud,
- BOD on *andmestruktuur*, mida kasutades võib töötada mõni teine programm (näiteks BOD-le vastava funktsiooni simuleerimiseks ehk väärtuse arvutamiseks);
- BOD on *viis esitada teavet* mingist digitaalskeemist (selle interpretatsiooni juurde tuleme edaspidi tagasi, vt. 1.2.4).

Teiselt poolt ei ole binaarne otsustusdiagramm muud kui võimalus esitada kombinatsioonskeemide funktsioone, käsitledes esitatavat skeemi nn. *musta kastina*. Skeemide diagnostiliseks modelleerimiseks oleks aga vaja käsitleda ka skeemi sisemist struktuuri s.t. simuleerida rikkeid skeemi sisepunktides. Soovides modelleerida BOD abil skeeme ventiilide tasandil diagnostika eesmärgil, oleks kaks võimalust:

- kujutada skeemi ventiilide võrguna ja esitada igat ventiili BOD kujul, või
- konstrueerida igale skeemi sisepunktile selle funktsiooni esitamiseks eraldi BOD.

Viimasel juhul on BOD puuduseks tema keerukuse eksponentsiaalne kasv kombinatsioonskeemide keerukuse kasvades.

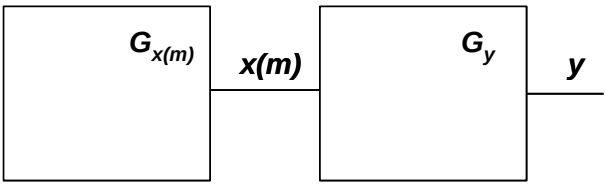
Järgnevalt vaatleme ühte BOD eriklassi, nn. *struktuurselt sünteesitud* BOD-sid (SSBOD), kus nimetatud puuduste mõju väheneb.

1.2.2. Struktuurselt sünteesitud binaarsed otsustusdiagrammid

Struktuurselt sünteesitud BOD mudel võeti kasutusele *ventiiltasandil* esitatud kombinatsioonskeemide diagnostiliseks modelleerimiseks signaaliteede terminites [Uba 76, Pla 80, Uba 96].

SSBOD sünteesi aluseks on binaarsete otsustusdiagrammide *superpositsiooni meetod* [Uba 76]. Olgu antud kaks kombinatsioonskeemi (joonis 1-15), millest üks on esitatud BOD-ga $G_y = (M_y, X_y, \Gamma)$, ja ühele tipule $m \in M_y$ vastav muutuja $x(m) \in X_y$ on teise kombinatsioonskeemi väljundiks, mis omakorda on esitatud ühe teise BOD-ga $G_{x(m)} = (M_{x(m)}, X_{x(m)}, \Gamma)$. Otsustusdiagrammide superpositsioon antud juhul seisneb

graafi G_y tipu $m \in M_y$ asendamises graafiga $G_{x(m)}$. Operatsioon seisneb järgmiste sammude läbiviimises.



Joonis 1-15. *Kaks omavahel ühendatud skeemi*

Algoritm 1-1. BOD superpositsioon:

1. Tipp m koos oma kahe kaarega eemaldatakse graafist G_y.
2. Terminaaltipud $m^{T,e}$, $e \in \{0,1\}$, eemaldatakse graafist $G_{x(m)}$.
3. Kõik terminaaltippudega $m^{T,e}$ seotud olnud kaared graafis $G_{x(m)}$ ühendatakse tipu $m \in M_y$ vastavate järglastega m^e graafis G_y.
4. Kõik tippu m sisenenud kaared graafis G_y ühendatakse graafi $G_{x(m)}$. *algtipuga* ehk juurega m_0.

Algoritmi 1-1 teel saadud graafe nimetamegi *struktuurselt sünteesitud binaarseteks otsustusdiagrammideks* (SSBOD).

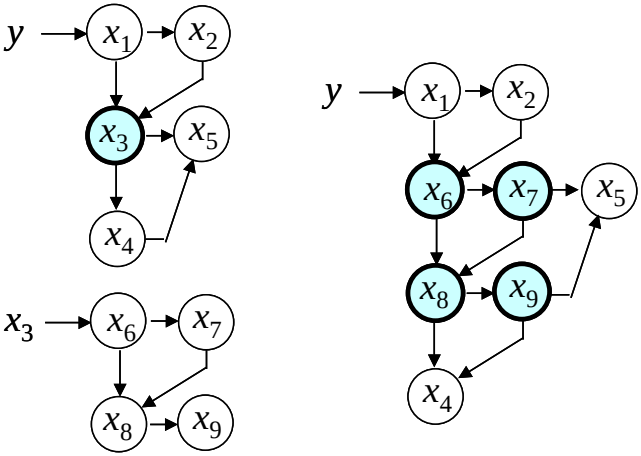
Näide 1-11.

Kahe BOD superpositsiooni operatsioon on illustreeritud joonisel 1-16. Funktsiooni $y = x_1x_2 \vee (x_3 \vee x_4)x_5$ esitava graafi G_y tipp m asendatakse teise graafiga $G_{x,3}$, mis esitab funktsiooni $x_3 = x_6x_7 \vee x_8x_9$. Uus graaf G_y esitab funktsiooni

$$y = x_1x_2 \vee (x_6x_7 \vee x_8x_9 \vee x_4)x_5.$$

Graafika lihtsustamiseks on joonisel 1-16 (ja edaspidi) terminaaltipud koos sinna suubuvate kaartega ära jäetud. Kuna kokkuleppeliselt kõik 1-kaared suunduvad alati paremale ja kõik 0-kaared alla, siis 1-kaarte puudumist tuleb interpreteerida, et graafist "väljumine" paremale tähendab

sisenemist 1-terminaali ja 0-kaarte kaarte puudumist tuleb interpreteerida, et graafist "väljumine" alla tähendab sisenemist 0-terminaali.

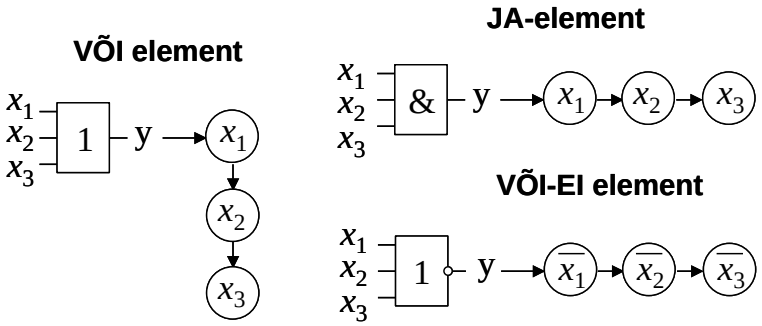


Joonis 1-16. *Binaarsete otsustusdiagrammide superpositsioon*

Kasutades BOD superpositsiooni võib suvalise ventiiltasemel esitatud kombinatsioonskeemile vastavusse seada struktuurselt sünteesitud BOD.

Olgu antud ventiiltasemel kombinatsioonskeem, kus iga ventiil (loogikaelement) on esitatud binaarse otsustusdiagrammiga. Niisuguste elementaardiagrammide näited on toodud joonisel 1-17.

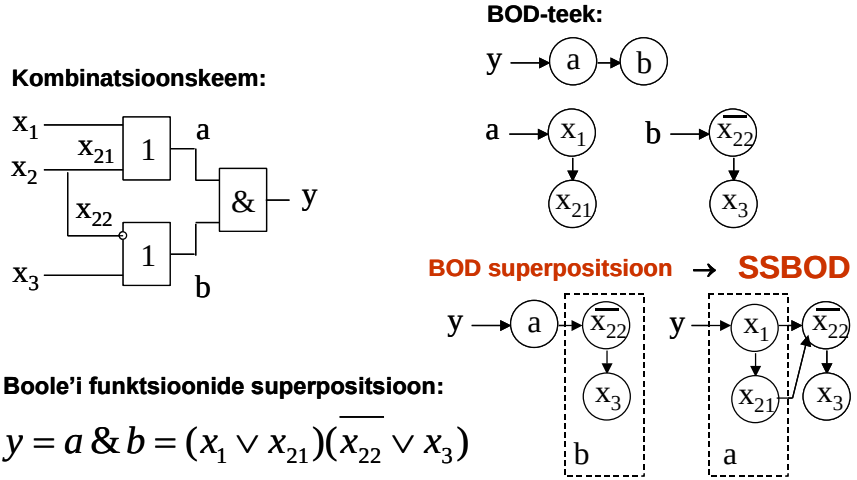
Alustades skeemi väljundelemendile vastavast graafist ja kasutades *iteratiivselt* superpositsiooni protseduuri saaksime graafide võrgu asemel üheainsa graafi (iga iteratsiooniga redutseerime mudelit ühe tipu ja ühe graafi võrra). Selleks, et redutseerida mudelit (tippude koguarvu) maksimaalselt, tuleks superponeerida graafe üksnes puukujuliste alamskeemide ulatustes. Sellisel juhul saaksime ventiilide võrgu asemel *puukujuliste alamskeemide* e. *makrode* võrgu, kus igale makrole vastab üks SSBOD.



Joonis 1-17. *Elementaasete binaarsete otsustusdiagrammide näited*

Näide 1-12.

Joonisel 1-18 on esitatud näide SSBOD genereerimisest puukujulisele kombinatsioonskeemile. Võrdluseks on toodud skeemile vastava Boole'i funktsiooni superpositsioon.



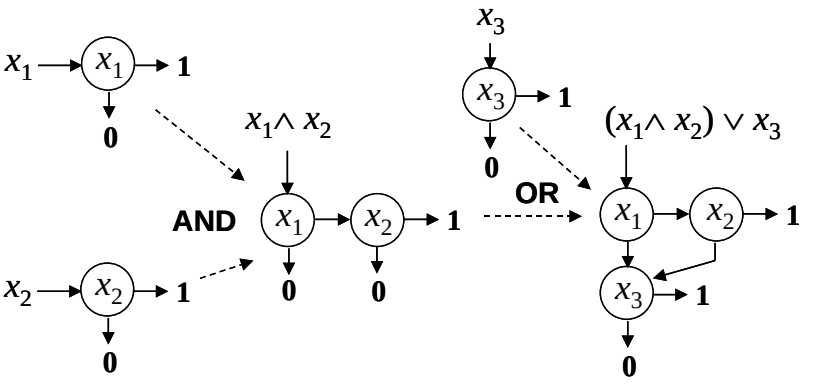
Joonis 1-18. *SSBOD mudeli süntees kombinatsioonskeemile*

SSBOD mudeli genereerimist antud kombinatsioonskeemile alustame joonisel 1-18 väljundelemendist JA ja sellele vastavast graafist G_y, mis koosneb kahest tipust *a* ja *b*. JA elemendi sisend *a* on samal ajal graafiga G_a esitatud VÕI elemendi väljundiks. Graaf G_a koosneb kahest tipust x_1 ja x_{21}. Kõigepealt asendame tipu *a* graafis G_y graafiga G_a, seejärel tipu *b* graafis G_y graafiga G_b, mis koosneb tippudest x_{22} ja x_3. Lõplik graaf, mis esindab antud puukujulist kombinatsioonskeemi ehk makrot, kust on välja jäetud hargnev sisend x_2, koosneb tippudest x_1, x_{21}, x_{22}, ja x_3.

Olgu märgitud, et samale ventiilskeemile saaks sünteesida SSBOD mudelit ka kasutades BOD *sümboolset manipuleerimist* [Bry 86]. Näide elementaarsete loogikaoperatsioonide kasutamisest BOD sünteesiks Boole´i funktsioonile $y = x_1x_2 \vee x_3$ või sellele vastavale kahest elemendist koosnevale ventiilskeemile alates sisenditest kuni väljundini [Sas 98] (vastupidiselt superpositsiooni protseduurile, mis toimub väljundist sisendite suunas) on esitatud joonisel 1-19.

Sümboolse manipuleerimise meetod graafidega loogikaoperatsioone sooritades on läbi viidav vaid ventiilide kaupa graafi ehitades. Superpositsioonimeetod on üldisem, kuna võimaldab suvalisi graafe (funktsioone) kokku ühendada.

SSBOD mudelit võib kasutada otseselt ventiiltasandil esitatud kombinatsioonskeemidele testide genereerimiseks.

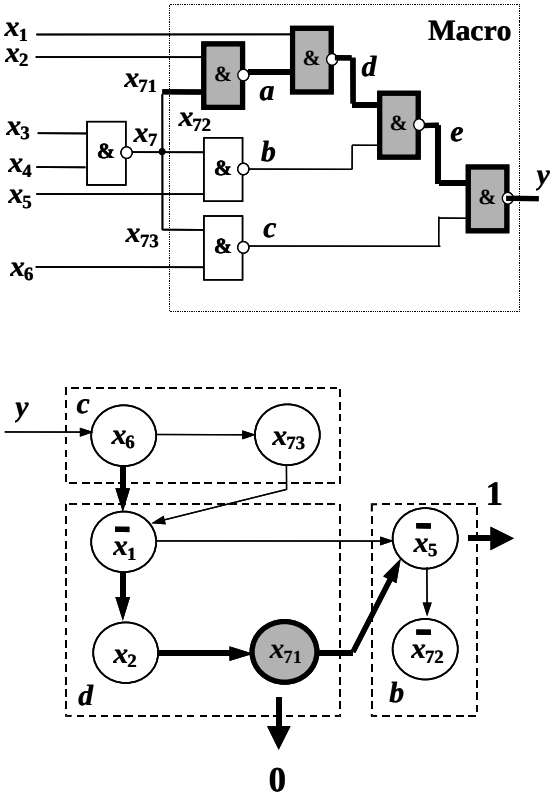


Joonis 1-19. *BOD mudeli süntees loogikaoperatsioone kasutades*

Erinevalt traditsioonilistest binaarsetest otustusdiagrammidest, kehtib SSBOD mudeli jaoks järgmine omadus: igale tipule m graafis G_y, mis kirjeldab mingis ventiiltasandil esitatud kombinatsioonskeemis N puukujulist alamskeemi N_y, vastab üheselt mingi *signaalitee* $l(m)$ skeemis N_y. See üksühene vastavus SSBOD tippude ja kombinatsioonskeemi signaaliteede vahel tuleneb otseselt SSBOD sünteesist superpositsiooni meetodil.

Näide 1-13.

Joomisel 1-20 on esitatud *puukujuline makro* ja talle vastav struktuurselt sünteesitud otsustusdiagramm.



Joonis 1-20. *Kombinatsioonskeem ja selle BOD mudel*

Igale tipule m tipumuutujaga $x(m)$ selles graafis joonisel 1-20 vastab signaalitee kombinatsioonskeemis, mis algab muutujaga $x(m)$ tähistatud ühendusest ja levib väljundini y. Tipumuutuja on inverteeritud, kui inversioonide arv talle vastaval signaaliteel on paaritu ja ilma inversioonita, kui paaris. Näiteks tipp muutujaga x_2 graafis esitab paarisarvu invertoritega signaaliteed läbi skeemipunktide x_2,a,d,e,y (rasvaselt tähistatud osa skeemis joonisel 1-20). Tipumuutuja x_1 on aga inverteeritud, kuna talle vastav signaalitee x_1,d,e,y sisaldab paaritut arvu invertoreid. Skeemipunktis x_7 on 3 hargnemist, kus iga haruga $x_{7,k}$ (k = 1,2,3) algavale ja väljundis y lõppevale signaaliteele skeemis vastab konkreetne sama muutujaga $x_{7,k}$ tähistatud tipp graafis.

Kirjeldatud SSBOD mudelil on rida eeliseid üldisema pelgalt Boole'i funktsioonide esitamiseks mõeldud BOD mudeliga võrreldes.

Esiteks, SSBOD modelleerib lisaks skeemi funktsioonile ka skeemi struktuuri – igale graafitipule vastab üks ja ainus signaalitee skeemis. Sellest tulenevalt on skeemi *struktuursete rikete* esitamine mudelis vahetu – skeemi rikkeid võib vaadelda graafitippude atribuutidena.

Teiseks, kombinatsioonskeeme võib diagnostika eesmärgil modelleerida SSBOD abil nii ventiilide kui ka kõrgemal makrode tasandil, kusjuures makroks võib olla suvaline osa skeemist (alamskeem). Erinevalt traditsioonilisest modelleerimisest ventiilide tasandil, kus iga ventiili tüübi esitamiseks on vajalik tema konkreetse mudeli hoidmine andmeteegis, SSBOD puhul ei ole selliste mudelteekide olemasolu vajalik. Kolmandaks, diagnostilise analüüsi meetodeid, mis on välja töötatud SSBOD mudeli baasil loogikatasandit silmas pidades, on võimalik vahetult üldistada kõrgema tasandi otsustusdiagrammide mudeli jaoks digitaalsüsteemide kõrgemaid esitustasandeid silmas pidades [Uba 81, Uba 83, Uba 96], vt. 1.2.4. – 1.2.7.

1.2.3. Diagnostikaoperatsioonid SSBOD mudelil

Vaatleme alljärgnevalt kahe põhilise diagnostikaülesande lahendamist struktuurselt sünteesitud otsustusdiagrammidel: pöördülesanne ehk testide genereerimine ja otseülesanne ehk rikete diagnoos.

Testide genereerimine

Olgu antud n sisendiga puukujuline skeem N_y, millele vastab mingi Boole'i funktsioon $y = F(X) = F(x_1,x_2,\ldots,x_n)$ ja SSBOD G_y. Genereerides

testi tipule m graafis G_y leitakse sisendsignaalide vektor, mis on testiks sisendile $x(m)$ skeemis N_y ja tervele reale riketele signaaliteel $l(m)$. Sama testi leidsime lahendades Boole'i differentsiaalvõrrandi (1-3).

Testi genereerimine SSBOD tipule toimub järgmise algoritmi kohaselt [Uba 96, Rai 98, Uba 98,].:

Algoritm 1-2. Testi genereerimine SSBOD tipule:

1. Aktiveerida tee l_m SSBOD algtipust tipuni m.
2. Aktiveeritakse kaks teega l_m mittevasturääkivat teed $l_{m,e}$, kus $e \in \{0,1\}$, tipu m naabritest m^e vastavate e-terminalideni $m^{T,e}$. Kui tipp m on vahetult ühendatud terminaliga $m^{T,e}$, ei ole konkreetse e väärtuse jaoks teed $l_{m,e}$ vaja aktiveerida.

Näide 1-14.

Vaatleme testi genereerimist graafi G_y tipule muutujaga x_{71} joonisel 1-20, millele vastab signaalitee skeemis alates punktist x_{71} läbi ventiilide a,d,e,y.

Aktiveerime *Algoritmi 1-2* kohaselt tee l_m läbi tippude[2] x_6, $\neg\, x_1$, x_2, valides vastavalt väärtused x_6=0, x_1=1 ja x_2=1. Tee $l_{m,1}$ aktiveerimiseks 1-terminali valime väärtuse x_5=0. Teed $l_{m,0}$ ei olegi vaja aktiveerida, kuna testitav tipp $x_{7,1}$ on juba ühendatud terminaliga 0. Lõplik testvektor on seega x_1,x_2,x_5,x_6,x_7 = 1100-. Kuna x_7 on skeemi sisepunkt, siis antud testi tuleb käsitleda kui makrokomponendi lokaalse sisendvektorina. Kuidas lõplikku testi saada antud haru testimiseks käsitleme edaspidi. Leitud testvektor aktiveerib skeemis rasvaselt näidatud signaalitee. Testi leidmiseks vajalikud aktiveeritud teed graafis on samuti näidatud rasvaselt.

Kui kasutaksime Boole'i differentsiaalaparaati antud ülesande lahendamiseks tuleks lahendada võrrand:

$$\frac{\partial y}{\partial x_{7,1}} = \frac{\partial(x_6 x_{7,3} \vee (\overline{x_1} \vee x_2 x_{7,1})(\overline{x_5} \vee \overline{x_{7,2}}))}{\partial x_{7,1}} =$$
$$= x_6 x_{7,3}(\overline{x_5} \vee \overline{x_{7,2}}) x_1 x_2 = x_1 x_2 \overline{x_5}\, \overline{x_6} \vee \ldots = 1$$

[2] Lihtsuse eesmärgil nimetame tippe m mõnikord ka neile vastavate tipumuutujate $x(m)$ järgi

Saaksime sama lahendi. Võrrandist nähtub, et lahendeid on veelgi rohkem, kuid kõik need sõltuvad muutuja x_7 väärtusest. Näiteks, üks lahenditest x_1,x_2,x_5,x_6,x_7 = 11-00 tähendab seda, et saaksime testida sisendis x_7 vaid signaalimuutusi $0 \to 1$, aga mitte muutusi $1 \to 0$. Selles mõttes on esimene lahend üldisem, kuna muutuja x_7 väärtusest mittesõltumine võimaldab testida sisendis x_7 mõlemasuunalisi signaalimuutusi.

Rikete diagnostika.

Olgu antud test vektor $T = (X^*,e)$, kus $e \in \{0,1\}$ on funktsiooni muutuja y väärtus sisendvektori X^* korral. Vektor X^* aktiveerib mingi tee $l(T)$ graafis terminaaltipuni $m^{T,e}$. Tipp $m \in l(T)$ sellel teel on siis ja ainult siis testitud kui lisaks teele $l(T)$ on aktiveeritud veel ka mingi teine tee $l_{m,\neg e}$ tipust $m^{\neg e}$ terminalini $m^{T,e}$.

Nimetatud tingimusest tuleneb järgmine *testvektorite diagnostilise analüüsi* algoritm.

Algoritm 1-3. Testvektori T diagnostiline analüüs SSBOD mudelil.

1. Simuleerida läbi testiga aktiveeritud tee $l(T)$ potentsiaalselt testitavate tippude leidmiseks. Olgu selle tee terminaaltipuks $m^{T,e}$.
2. Iga tipu $m \in l(T)$ jaoks jätkata simuleerimist alates selle tipu järglasest $m^{\neg e} \notin l(T)$, et kontrollida, kas teine tee $l_{m,\neg e}$ terminaaltippu $m^{T,\neg e}$ on samuti aktiveeritud; kui on, siis antud vektor T testib tippu m.

Näide 1-15.

Vaatleme veelkord joonist 1-20. Lokaalne testvektor joonisel esitatud makro jaoks T = (11001,1) aktiveerib graafis tee $l(T) = (x_6,\neg x_1, x_2, x_{7,1},\neg x_5, m^{T,1})$. Järgnevalt kontrollime aktiveeritud tee $l_{m,0}$ olemasolu tippudele sellel teel $l(T)$. Tippudel x_6 ja $\neg x_1$ see puudub, mõlema tipu puhul tipumuutuja väärtuse muutumine ei muuda funktsiooni y väärtust. Tippude jaoks x_2 ja $x_{7,1}$ ei olegi vaja seda kontrolli läbi teha – ühendus 0-terminaliga on vahetu. Tipu $\neg x_5$ jaoks tee $l_{m,0}$ samuti puudub. Seega vektor T testib antud graafis üksnes tippe x_2 ja $x_{7,1}$ ehk sellele vastavalt mõlemat signaaliteed skeemis alates sisenditest x_2 ja $x_{7,1}$ läbi ventiilide a,d,e ja y.

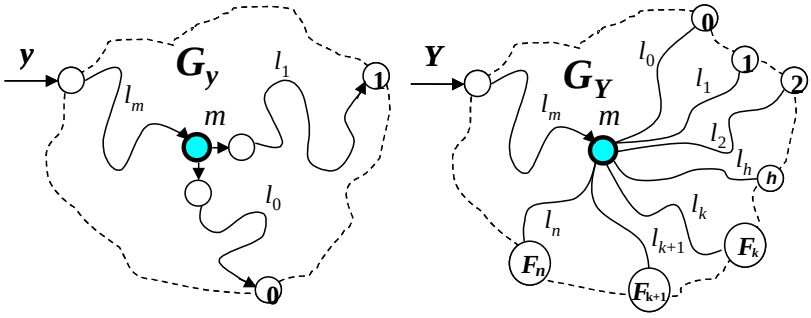
Kui testvektorit rakendada makro sisenditele ja väljundsignaali *y* väärtus osutub vigaseks tuleb kahtlustada rikkeid tippudes x_2 või $x_{7,1}$ ehk neile vastavatel signaaliteedel skeemis.

Vaadeldud operatsioonide algoritmid põhinevad SSBOD mudeli topoloogilisel töötlusel ja seetõttu võib siin Boole'i algebra konteksti kõrvale jätta.

Omistame aktiveeritud teele graafis põhimõtteliselt uudse tähenduse: selle asemel, et käsitleda aktiveeritud teed kui Boole'i differentsiaalvõrrandi lahendit ehk ühte termi disjunktiivses või konjunktiivses normaalkujus, võime teda vaadelda kui tagajärg-põhjuse (testide genereerimine) või põhjus-tagajärje (rikete diagnostika) funktsiooni.

Kui tahame kavandada testi mingite objektide (graafitippude) käitumise kontrollimiseks, siis tuleb konstrueerida niisugune topoloogiline situatsioon graafis, kus graafiga esitatava funktsiooni väärtus hakkab sõltuma sellest, millises suunas testitavast tipust väljutakse.

Binaarses otsustusdiagrammis G_y (joonis 1-21a) tuleb selleks aktiveerida kolm teed (nn. "kolmjalg"): tee l_m graafi algusest testitavasse tippu *m*, ja tipu *m* naabritest kaks teed l_0 ja l_1 vastavalt 0-terminali ja 1-terminali. Sõltuvalt tipust *m* väljumise suunast, same funktsioonile erinevad väärtused.



a) b)

Joonis 1-21. *Binaarne ja kõrgtaseme otsustusdiagramm*

Üldistades nüüd otsustusdiagrammi mõistet kõrgtaseme otsustusdiagrammiks G_Y (joonis 1-21), lubame muutujatele $x(m)$ rohkem

väärtusi kui 0 või 1 ning laseme ka terminaltippude arvu vabaks. Terminalides võivad, kuid ei pea enam olema konstandid 0,1,2,…,h,…,k-1, vaid seal võivad olla ka suvalised funktsionaalsed avaldised F_k, F_{k+1},…,F_n. Tipu m testimise ülesanne topoloogilises mõttes jab aga ikka samaks: on vaja aktiviseerida algtee l_m ja abiteed l_0, l_1, …, l_{k-1}, l_k, l_{k+1},…,l_n tipu m naabritest vastavalt terminalidesse 0,1,…,k-1, F_k, F_{k+1},…,F_n.

1.2.4. Otsustusdiagrammide üldjuhtum

Olgu *digitaalsüsteem* kirjeldatav funktsioonina $Y = F(X) = F(X_1, X_2,..., X_n)$, kus $X_j \in X$ on täisarvulised muutujad *määramispiirkondadega* $V(X_j)$, ning funktsioon F on esitatatav kujul F: $V(X_1) * V(X_2) *...* V(X_n) \rightarrow V(Y)$, kus $V(Y)$ - on funktsiooni *muutumispiirkond* ja * - tähistab *Descartes'i korrutist*. Defineerime digitaalsüsteemi $y = F(X)$ esitavat otsustusdiagrammi (OD) üldjuhul kui *orienteeritud tsükliteta graafi* $G_Y = (M, X, \Gamma)$, kus $M = \{m\}$ on *tippude* hulk, $X = \{X_1,X_2,…,X_n\}$ on funktsiooni argumentide hulk, $X(m) \in X$ on muutuja, mis on vastavusse seotud tipuga m, ning $\Gamma(M,X)$ on funktsiooniga $Y = F(X)$ üheselt määratud relatsioon, kus $\Gamma(m,X(m))$ määrab tipu m järglase vastavalt *tipumuutuja* $X(m)$ määramispiirkonnale $V(X(m))$. Vektoriga X^* on graafis määratud liikumine tipust tippu. Igal tipul m on $|V(X(m))|$ väljuvat kaart ning vastavalt $k \leq |V(X(m))|$ naabrit m^e, $e \in V(X(m))$. Graafis on N *terminaaltippu* $m^{T,i}$, $i = 1,2,… N$, $N \geq 2$, märgenditega $x(m^{T,i})$, mis võivad olla konstandid, muutujad $X_j \in X$ või funktsionaalsed avaldised muutujatest $X_j \in X$.

Me ütleme, et tipumuutuja $X(m)$ väärtus aktiveerib mingi tipu m väljundkaare. Mingi tee on graafis aktiveeritud siis, kui kõik seda teed moodustavad kaared on aktiveeritud. OD on aktiveeritud väärtusele $e \in V(Y)$ siis kui graafis on aktiveeritud tee algtipust (juurest) mingi terminalini m^T, nii et $X(m^T) = e$. Otsustusdiagramm $G_Y = (M, X, \Gamma)$ esitab funktsiooni $Y = F(X)$, siis ja ainult siis, kui iga vektori X^* jaoks, mis viib graafis terminaaltippu m^T, kehtib $F(X^*) = X(m^T)$.

Sõltuvalt digitaalsüsteemi klassist või süsteemi esitustasemest, me võime klassifitseerida eri tüüpi otsustusdiagramme erinevate muutujate ja tippude interpretatsioonidega.

Vaatleme kahte otsustusdiagrammide sünteesi meetodit: süsteemi *protseduurse mudeli sümbolsimuleerimine* ja süsteemi struktuurse OD võrgu superpositsioon. Viimasel juhul süsteemi komponentide OD-mudelid peavad eelnevalt juba sünteesitud olema.

OD süntees süsteemi protseduurse mudeli sümbolsimuleerimise abil

Olgu antud mingi süsteemi protseduurne kirjeldus. Sümbolsimuleerimise teel genereeritakse puu protseduuri kõigist võimalikest teedest (süsteemi *tööviisidest*). Iga tee jaoks leitakse kõigi vajalike tingimuste hulk *C*, milliste korral antud tee realiseerub ja tegevuste (operatsioonide) hulk *P*, mis antud tingimustel realiseeruvad. Hulgad *C* ja *P* on aluseks OD-mudeli konstrueerimisel.

```
BEGIN
    Memory state:       M
    Processor state:    PC, AC, AX
    Internal state:     TMP
    Instruction format: IR = OP. A. F0. F1. F2.
    Execution process: EXEC:
                       BEGIN
                       DECODE  OP ⇒ (
                               0:  AC← AC + M? A?
                               1:  M[A]← AC,   AC ← 0
                               2:  M[A]← M[A]+ 1,
                                   IF  M[A]= 0 THEN PC ← PC + 1
                               3:  PC ← A
                               .....................................
                               7:  IF F0 THEN  AC ← AC + 1
                                   IF F1 THEN  IF AC = 0  THEN  PC ← PC + 1
                                   IF F2 THEN  (TMP ← AC, AC ← AX, AX ← TM')
                       END
END
```

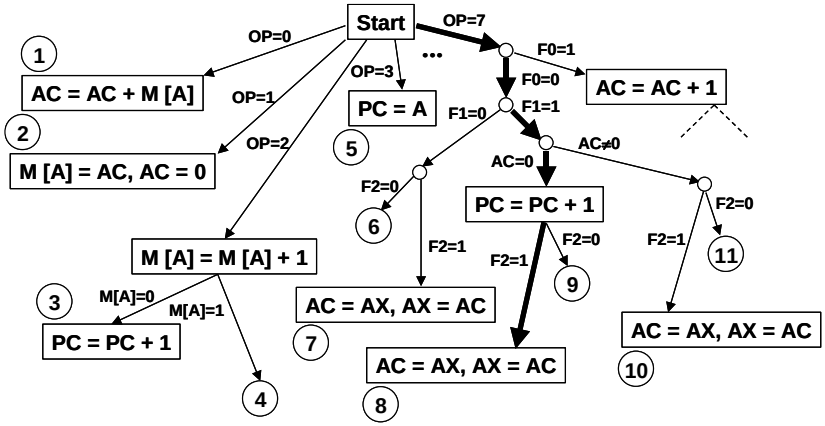
Joonis 1-22. *Fragment protsesssori protseduursest kirjeldusest*

Vaatleme näitena fragmenti ühe *protsessori protseduursest kirjeldusest* joonisel 1-22.

Protsessori andmemuutujateks on *M*, *PC*, *AC* ja *AX*, *juhtmuutujateks* on *käsuformaadi* väljad: *operatsioonikood OP* ja *loogikatingimused* (lipud) *F*0, *F*1 ja *F*2. Täisarvulise muutuja *OP*, mille määramispiirkonnaks on $V(OP) = \{0,1,2, \ldots,7\}$ ja *binaarmuutujate F*0, *F*1 ja *F*2 kõigi väärtuste läbi simuleerimisel genereeritaks puu joonisel 1-23.

Iga haru selles puus kujutab ühte võimaliku teed läbi protseduuri joonisel 1-22. Puu kaartele on märgitud tingimused hulgast *C*, milliste puhul antud kaar realiseerub, puu tippudes on aga esitatud operatsioonid hulgast P, mis vastava tee läbimisel täide viiakse. Sümbolsimuleerimise puu alusel koostatakse loetelu kõikvõimalikest paaridest - tingimuste

alamhulgast ja operatsioonidest (süsteemi töörezhiimidest), mis antud tingimustel toimuvad.



Joonis 1-23. *Sümbolsimuleerimise puu*

Näide niisugusest loetelust joonisel 1-23 toodud puu jaoks on esitatud Tabelis 1-2.

Tabel 1-2 *Nimistu süsteemi töörezhiimidest*

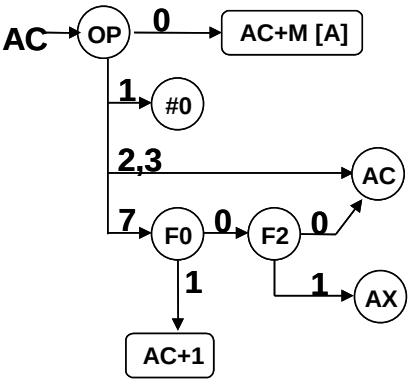
No	Input assertions	Output assertions
1	OP = 0	AC = AC+ M [A]
2	OP = 1	M [A] = AC, AC = 0
3	OP = 2, M [A] + 1 = 0	M [A] = M [A] +1, PC = PC + 1
4	OP = 2, M [A] + 1 ≠ 0	M [A] = M [A] + 1
5	OP = 3	PC = A
6	OP = 7, FO=0, F1=0, F2=0	NO CHANGE
7	OP = 7, FO=0, F1=0, F2=1	AC = AX, AX = AC
8	OP = 7, FO=0, F1=1, AC=0, F2=1	AC = AX, AX = AC
9	OP = 7, FO=0, F1=1, AC=0, F2=0	NO CHANGE

Tabeli paremal poolel esitatud operatsioonide hulgast ekstrahheerime *funktsiooni muutujad* (avaldiste vasakud pooled) *Y = {AC, AX, M, PC}*. Nende muutujate jaoks on võimalik konstrueerida otsustusdiagrammid, kusjuures tingimustega Tabeli 1-2 vasakust poolest märgendame OD *mitteterminaalseid* tippe ja operatsiooniavaldiste paremate pooltega tabeli

paremast poolest märgendame OD terminaaltipud. Nii näiteks muutuja *AC* jaoks, viies läbi vajalikud lihtsustused, leiame järgmise sõltuvuse:

$$AC = (OP = 0)\,(AC + M[A]) \vee (OP = 1)\,(0) \vee [(OP = 2) \vee (OP = 3)]\,AC \vee$$
$$\vee\,(OP = 7)\,[(F0 = 0)\,((F2 = 0)\,AC \vee (F2 = 1)\,AX) \vee$$
$$\vee\ (F0 = 1)\,(AC + 1)],$$

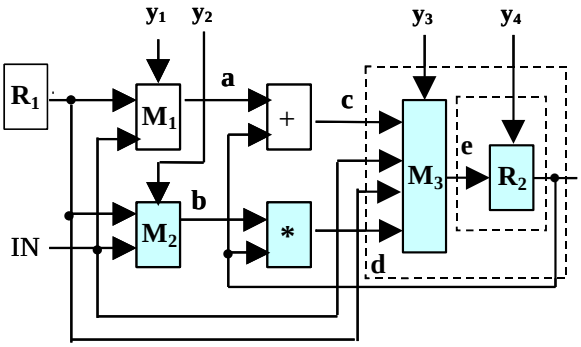
millele vastab otsustusdiagramm joonisel 1-24.



Joonis 1-24. *Otsustusdiagramm funktsiooni muutuja AC jaoks*

OD süntees superpositsiooni meetodil

Joonisel (1-25) on esitatud digitaalsüsteemi *operatsioonautomaat*, mis koosneb järgmistest *operatsioonelementidest*: registritest R_1 ja R_2, kolmest multipleksorist M_1, M_2, M_3, summaatorist ja korrutajast. Süsteemi juhtmuutujateks on sisendsignaalid y_1, y_2, y_3, y_4, ning andmesisendiks sõnamuutuja *IN*. Komponentide M_1, M_2, M_3 ja R_2 funktsioonid on esitatud Tabelis 1-3. Nendele funktsioonidele vastavad otsustusdiagrammid $G_{M,1}$, $G_{M,2}$, $G_{M,3}$, $G_{R,2}$, on toodud joonisel 1-26.



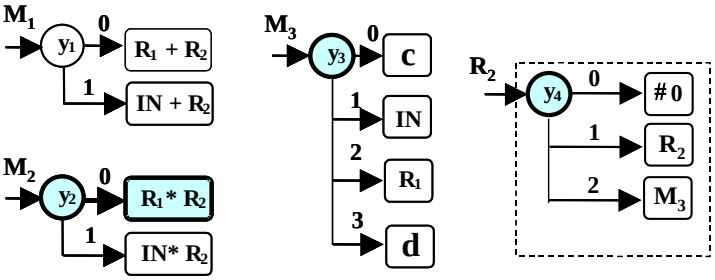
Joonis 1-25. *Digitaalsüsteemi operatsioonautomaat*

Tabel 1-3 *Joonisel 1-25 esitatud operatsioonelementide funktsioonid*

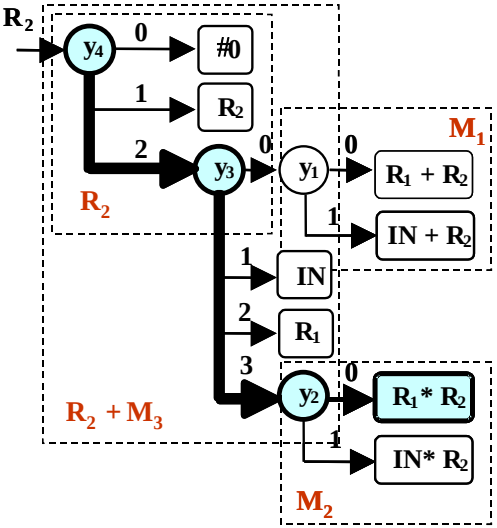
M_1		M_2		M_3		R_2		
y_1	Function	y_2	Function	y_3	Function	y_4	Operation	F
0	$M_1 = R_1$	0	$M_2 = R_1$	0	$M_3 = M_1 + R_2$	0	Reset	$R_2 = 0$
1	$M_1 = IN$	1	$M_2 = IN$	1	$M_3 = IN$	1	Hold	$R_2 = R'_2$
				2	$M_3 = R_1$	2	Load	$R_2 = M_3$
				3	$M_3 = M_2 * R_2$			

Viies läbi järgnevalt superpositsiooni protseduuri, kus kõigepealt asendades graafis $G_{R,2}$ tipu M_3 graafiga $G_{M,3}$, seejärel saadud graafis tipu c graafiga $G_{M,1}$ ja tipu d graafiga $G_{M,2}$ saame uue superponeeritud graafi $G_{R,2}$ joonisel 1-27, mis esitab kompaktselt operatsioonautomaati joonisel 1-25.

Otsustusdiagrammi $G_{R,2}$ eeliseks võrreldes traditsioonilise komponentide võrguga joonisel 1-25 on effektiivsuse tõus süsteemi simuleerimisel ja diagnostilisel modelleerimisel. Näiteks selle asemel, et juhtvektori y_1, y_2, y_3, y_4 = 0032 simuleerimiseks läbi viia lähteskeemis joonisel 1-25 arvutusi $a = R_1$, $b = R_1$, $c = a + R_1$, $d = b * R_1$, e = d, ja $R_2 = e$, tuleks OD-mudelil $G_{R,2}$ joonisel 1-27 läbi käia aktiveeritud tee läbi tippude y_4, y_3 ja y_2 terminaltippu, kus oleks vaja sooritada üksainus tehe $R_2 = R_1 * R_2$.



Joonis 1-26. *OD-d süsteemi komponentide jaoks joonisel 1-25*



Joonis 1-27. *Otsustusdiagramm operatsioonautomaadi jaoks joonisel 1-25*

Diagnostilise analüüsi effektiivsus otsustusdiagrammidel tõuseb sellest, et tagajärg-põhjus funktsionaalsed suhted on siin paremini korrastatud ning järeldused tulenevad vahetult. Näiteks väära signaali avastamisel testvektori y_1, y_2, y_3, y_4 = 0032 puhul operatsioonautomaadi

väljundis tulenevad võimalikud veapõhjused otseselt aktiveeritud teel läbitud tippudest.

Testide genereerimise efektiivsus tõuseb OD-mudelit kasutades antud juhul sellest, et tänu operatsioonautomaadi alamfunktsioonide superponeerimisele, õnnestub testi genereerida vahetult sisendmuutujate peal, ilma et oleks vaja läbi viia täiendavaid sisemuutujate väärtuste arvutamisi.

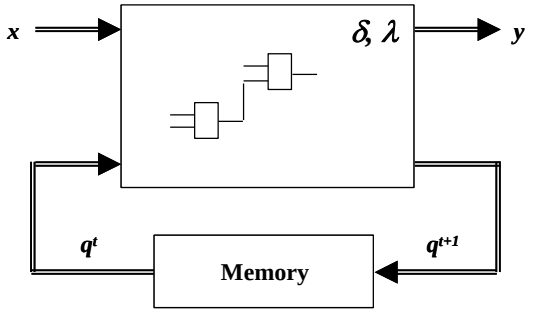
Näiteks, selleks et sünteesida testi korrutajale joonisel 1-25 toodud skeemi abil, oleks vaja levitada nii stiimuleid alates skeemisisenditest läbi skeemikomponentide testitava korrutaja sisenditeni kui ka transportida korrutaja reaktsioone läbi skeemikomponentide väljundregistrini R_2. Kasutades aga OD-mudelit on vaja üksnes leida terminaaltipp korrutaja funktsiooniga ja aktiveerida graafil tee, mis viib sellesse tippu.

1.2.5. Otsustusdiagrammid juhtautomaatidele

Digitaalsüsteemide *juhtautomaate* võib esitada nii struktuuri kui ka funktsioonide kaudu. Esimesel juhul vaatleme automaati *Hufmann'i* mudelina (joonis 1-28) [Gri 99], kus automaat on dekomponeeritud kombinatsioonskeemiks ja mäluskeemiks, ning teisel juhul vaatleme automaati väljundite ja üleminekute diagrammina.

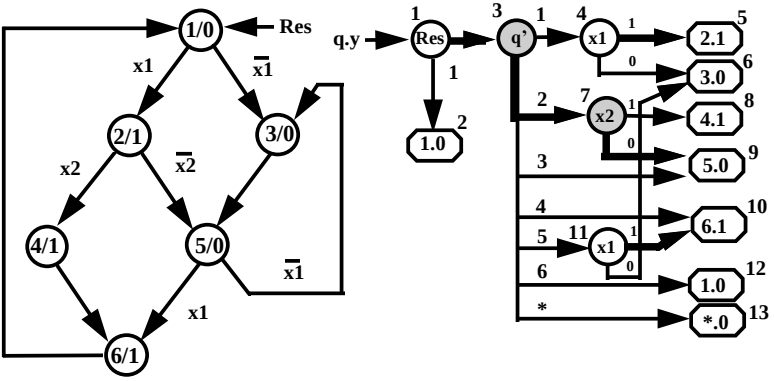
Struktuurse käsitlusviisi puhul, kus automaadi kombinatsioonskeemi osa, mis realiseerib *väljundfunktsioone* λ ja *siirdefunktsioone* δ, esitatakse ventiilide võrguna, võime kasutada SSBOD mudelit juba vaadeldud kujul. Testide genereerimise efektiivsuse tõstmist silmas pidades aga pole automaatide struktuursest käsitlemisest ventiilide tasandil kasu ülesande keerukuse tõttu. Lõplike automaatide diagnostilise analüüsi efektiivsuse tõstmiseks oleks vaja neid käsitleda just kõrgematel funktsionaalsetel tasanditel.

Jättes lahtiseks automaadi olekute ja sisend/väljundmuutujate kodeerimise võime esitada automaadi sisend-, väljund- ja olekumuutujad täisarvulistena, mis võimaldab automaadi funktsioonide esitamisel loobuda keerukast loogikatasandist.



Joonis 1-28 *Lõpliku automaadi Hufmann'i mudel*

Piirdudes praegu üksnes *Moore'i automaadi* käsitlemisega võime nii väljundmuutuja *y* kui ka olekumuutuja *q* esitada *konkatenatsiooni* abil üheainsa *liitmuutujana q.y* ja konstrueerida automaadi *integreeritud funktsiooni* (*vektorfunktsiooni*) kujul $q.y = f(q',x)$, kus *q'* tähistab automaadi eelmist olekut. Niisuguse funktsiooni saaks esitada üheainsa *vektorotsustusdiagrammina* (joonis 1-29).



Joonis 1-29 *Lõpliku automaadi üleminekute-väljundite graaf ja otsustusdiagramm*

Näide 1-16.

Joonisel 1-29 on esitatud juhtautomaadi olekute ja üleminekute ühitatud diagramm ja OD-mudel. Otsustusdiagramm esitab automaadi liit- ehk vektorfunktsiooni $q.y = F(q', Res, x_1, x_2)$, kus $q.y$ – on konkatenatsioon täisarvulisest muutujast q määramispiirkonnaga $V(q) = \{1,2,3,4,5,6\}$ ja väljundmuutujast y, mis on binaarne, *Res* (reset) – on automaadi binaarne *seademuutuja* (*Res* = 1 viib automaadi algolekusse $q = 0$) ja x_1 ning x_2 on automaadi binaarsed sisendmuutujad. Apostroofiga (q') tähistame automaadi olekut eelmisel taktil. Terminaaltipud on mudelis märgendatud *liitkonstantidega* – liitmuutuja $q.y$ võimalike väärtustega (vastavalt, järgmine olek ja väljundi y väärtus). Võimalike rikete tõttu tekkida võiva "*illegaalse*" oleku tähistamiseks kasutame tärni * (mudelis eeldame, et illegaalse oleku puhul on automaadi väljundmuutuja y väärtuseks 0).

Võib vaadelda kahte äärmuslikku juhtu lõplike automaatide funktsionaalsel esitusel:

1) kasutame automaadi modelleerimiseks vaid kolme täisarvulist abstraktset muutujat sisendi, väljundi ja olekute esitamiseks ning vastavalt sellele ka vaid kahte (või ühte integreeritud) otsustusdiagrammi üleminekute (siirete) ja väljundkäitumise kirjeldamiseks;

2) automaadi sisendid, väljundid ja olekud kodeeritakse; sellisel juhul võib automaati esitada binaarfunktsioonide süsteemina.

Näide 1-17.

Vaatleme otsustusdiagrammide sünteesi juhtautomaadile *VHDL kirjeldusest*. Joonisel 1-30 on esitatud 4 *protsessist* koosnev juhtautomaadi VHDL kirjeldus. Iga protsessi väljundmuutuja käitumise diagnostiliseks modelleerimiseks sünteesime sellele protsessile vastava otsustusdiagrammi *sümbolsimuleerimise* meetodil.

Järgnevate lihtsustuste läbiviimiseks võtame kasutusele järgmise tähistuse. Olgu

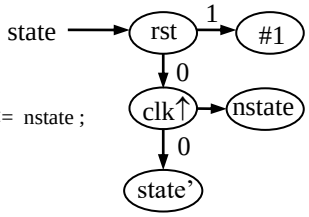
$$CL = \{c\uparrow, c, c\downarrow, \neg c\}$$

loogikamuutujate hulk tähistamaks erinevaid tüüpe taktsignaaliga seotud protsesside aktiveerimistingimusi.

```
entity rd_pc is
  port
    ( clk, rst : in bit; rb0 : in bit; enable : in bit; reg_cp : out bit ;
    reg : out bit ; outreg : out bit ; fin : out bit ) ;
end rd_pc ;
architecture archi_rd_pc of rd_pc is type STATETYPE is
  (state1, state2);
  signal state, nstate: STATETYPE ; signal enable_in : bit ;
  signal reg_cp_comb : bit ;
begin
  seq: process(clk, rst)
  begin
    if rst='1' then state <= state1 ;
    elsif (clk'event and clk='1') then state <=  nstate ;
    end if ;
  end process ;
  process(clk, enable)
  begin
    if clk='1' then   enable_in <= enable ;
    end if ;
  end process ;
  process(clk, reg_cp_comb)
  begin
    if clk='0' then   reg_cp <= reg_cp_comb ;
    end if ;
  end process ;
  comb: process (state, rb0, enable_in)
  begin
    case state is
      when state1 => outreg <= '0' ; fin <= '0' ;
        if (enable_in='0') then   nstate <= state1 ;
                          reg <= '1' ; reg_cp_comb <= '0' ;
                     else  nstate <= state2 ; reg <= '1' ;
                          reg_cp_comb <= '1' ;
        end if ;
      when state2 =>
        if (rb0='1') then   nstate <= state2 ; reg <= '0' ;
                          reg_cp_comb <= '1'; outreg <= '0'; fin <= '0';
          elsif (enable_in='0') then   nstate <= state1 ; reg <= '0' ;
                          reg_cp_comb <= '0'; outreg <= '1'; fin <= '1';
                          else  nstate <= state2 ; reg <= '0' ;
                          reg_cp_comb <= '0'; outreg <= '0'; fin <= '1';
        end if ;
    end case ;
  end process ;
end archi_rd_pc ;
```

Joonis 1-30. *Juhtautomaadi VHDL kirjeldus*

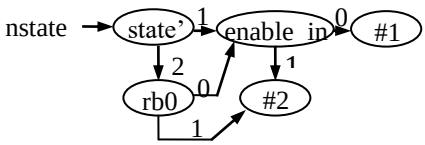
```
process(clk, rst)
begin
  if rst='1' then state <= state1 ;
  elsif (clk'event and clk='1') then state <=  nstate ;
  end if ;
end process ;
```



.

Joonis 1-31. *VHDL protsessi "state" otsustusdiagramm*

```
comb: process (state, rb0, enable_in)
begin
  case state is
    when state1 => outreg <= '0' ; fin <= '0' ;
      if (enable_in='0') then   nstate <= state1 ;
                          reg <= '1' ; reg_cp_comb <= '0' ;
                    else  nstate <= state2 ; reg <= '1' ;
                          reg_cp_comb <= '1' ;
      end if ;
    when state2 =>
      if (rb0='1') then   nstate <= state2 ; reg <= '0' ;
                    reg_cp_comb <= '1'; outreg <= '0'; fin <= '0';
        elsif (enable_in='0') then   nstate <= state1 ; reg <= '0' ;
                    reg_cp_comb <= '0'; outreg <= '1'; fin <= '1';
                    else  nstate <= state2 ; reg <= '0' ;
                    reg_cp_comb <= '0'; outreg <= '0'; fin <= '1';
      end if ;
  end case ;
end process ;
```



Joonis 1-32. *VHDL protsessi "nstate" otsustusdiagramm*

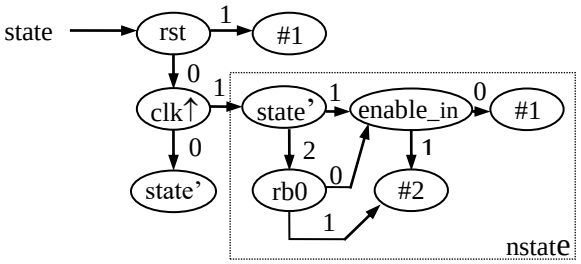
VHDL kirjelduse kontekstis tähendab $c\uparrow$, et protsess on aktiveeritud, kui *clk'event* & *clk* ='1' on tõesed, c - tähendab, et protsess on aktiveeritud, kui *clk* = '1' on tõene, $c\downarrow$ - tähendab,et protsess on aktiveeritud, kui *clk'event* & *clk* ='0' on tõene, ja $\neg c$ tähendab, et protsess on aktiveeritud, kui *clk* = '0' on tõene. Nagu varemgi, apostroof muutuja juures tähendab, et muutuja väärtust tuleb võtta eelneval ajahetkel. Defineerime ka järjestuse:

$$c\uparrow < c < c\downarrow < \neg c$$

määramaks, millises järjekorras protsessid ühe ja sama taktperioodi vältel aktiveeritakse

VHDL protsessidele *state* ja *nstate* vastavad otsustusdiagrammid G_{state} ja G_{nstate} on esitatud vastavalt joonistel 1-31 ja 1-32. DD sünteesiks on kasutatud peatükis 1.2.4 vaadeldud sümbolistlikku simuleerimise meetodit. Graafi G_{nstate} sünteesil on seejuures simuleeritud üksnes muutuja *nstate* käitumist. Ülejäänud väljundmuutujatele *reg,, reg_cp, outreg* ja *fin* tuleks sünteesida nendele vastavad otsustusdiagrammid.

Kahest graafist G_{state} (joonis 1-31) ja G_{nstate} (joonis 1-32) koosnevat DD-mudelit saab lihtsustada, viies läbi graafide *superpositsiooni*. Selleks asendame tipu *nstate* graafis G_{state} graafiga G_{nstate}. Uus superponeeritud graaf on toodud joonisel 1-33, kusjuures iseseisev graaf G_{nstate} on nüüd mudelist kadunud, olles graafi G_{state} alamgraaf.

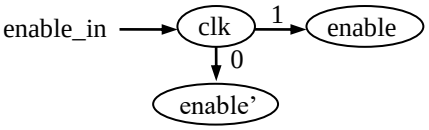


Joonis 1-33. *Protsessi "state" otsustusdiagramm pärast superpositsiooni*

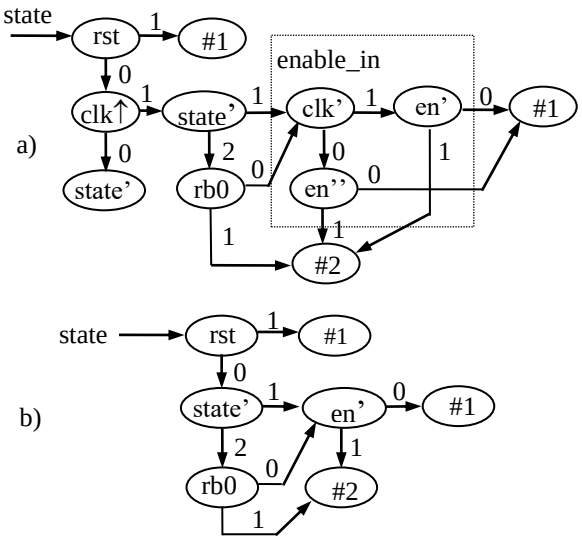
Joonisel 1-34 on toodud VHDL protsessi *enable_in otsustusdiagramm* G_{enable_in}. Kuna muutuja *enable_in* on tipumuutujaks graafis G_{state}, siis on nüüd võimalik veelgi mudelit lihtsustada asendades tipu *enable_in* graafis G_{state} graafiga G_{enable_in}. Vastav uus graaf G_{state} on näidatud joonisel 1-35a.

Võttes arvesse taktsignaaliga seotud muutujate ajalist järgnevust $c\uparrow < c < c\downarrow < \neg c$ ning ka seda, et taktsignaali ennast ei pea sünkroonskeemide mudelis ilmtingimata otseselt esitama [Lev98], võime läbi viia täiendavad lihtsustused ja esitama otsustusdiagrammi G_{state} veelgi kompaktsemalt joonisel 1-35b.

```
process(clk, enable)
begin
  if clk='1' then   enable_in <= enable ;
  end if ;
end process ;
```

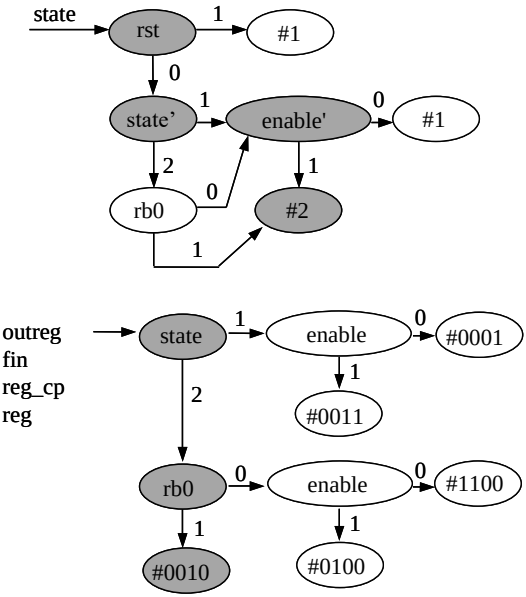


Joonis 1-34. *Protsessi "enable" otsustusdiagramm*



Joonis 1-35. Tipu "enable_in" superpositsioon graafis G_{state}

Superpositsiooni tulemusena on nüüd osa signaale esialgsest VHDL mudelist (joonis 1-30), mille kaudu protsessid kommunikeeruvad, ja ka vastavatest originaalotsustusdiagrammidest elimineeritud. Nii näiteks on elimineeritud graafist G_{state} (joonis 1-31) muutujad *clk*↑ ja *nstate*, graafist G_{nstate} (joonis 1-32) *enable_in* ja graafist G_{enable_in} (joonis 1-34) *clk'* ja *en*''. Lihtsustuste tulemusena on esialgse VHDL mudeli üleminekufunktsioonide esitus graafina G_{state} joonisel 1-35 väga kompaktne. See omakorda tõstab modelleerimise kiirust ja vähendab otsinguruumi mahtu testide genereerimisel.



Joonis 1-36. *Juhtautomaadi VHDL kirjelduse lõplik esitus OD mudelina.*

Joonisel 1-36 on esitatud VHDL keeles kirjeldatud juhtautomaadi lõplik OD-mudel kahe graafina. Esimene graaf G_{state} esitab automaadi üleminekute funktsiooni, teine graaf väljundite funktsiooni. Väljundmuutuja on vektoriaalne, tema argumentideks on binaarmuutujad *outreg, fin, reg_cp ja reg.* Kasutades vektormuutujat on võimalik väljundfunktsiooni mudelit esitada kompaktselt üheainsa vektorgraafina.

OD-mudel võimaldab lihtsustada testide genereerimist võrreldes esialgse VHDL kirjeldusega, kuna kogu funktsionaalses mõttes "ebaoluline info" on OD-mudelist eemaldatud. Käies OD-mudelis joonisel 1-36 läbi kõik teed ja aktiveerides neid saaksime genereerida vaid 6-st vektorist koosneva funktsionaalse testjada, mis on ära toodud Tabelis 1-4. Tabelis ära värvitud testvektorile vastavad aktiveeritud teed OD-mudelis joonisel 1-36 on samuti värviga esile toodud.

Tabel 1-4 *Testide genereerimine OD-mudelil joonisel 1-30 esitatud automaadile*

time	1	2	3	4	5	6
rst	1	0	0	0	0	0
enable	0	1	1	1	0	0
rb0	x	x	1	0	0	0
state	1	1	2	2	2	1
outreg	0	0	0	0	1	0
fin	0	0	0	1	1	0
reg_cp	0	1	1	0	0	0
reg	1	1	0	0	0	1

1.2.6. Otsustusdiagrammid digitaalsüsteemidele registersiirete tasandil

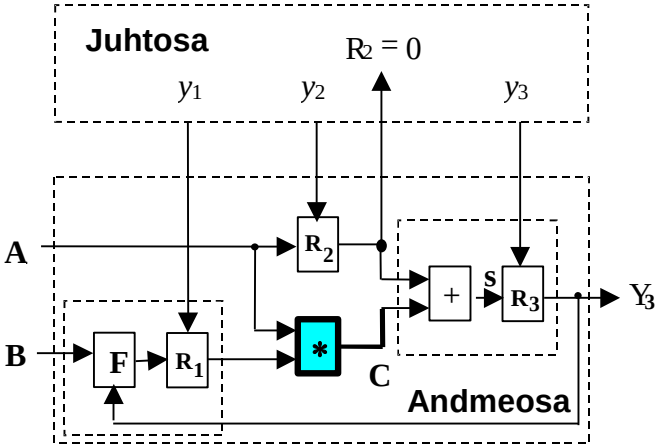
Digitaalsüsteemide kirjeldamiseks OD mudeli abil *registersiirete tasandil* tuleb kõigepealt esitada süsteem sobiva komponentide või alamsüsteemide võrguna ja seejärel kirjeldada alamsüsteemide funktsioone otsustusdiagrammidena. Alamsüsteemiks on otstarbekas valida register koos oma sisendloogikaga, niiviisi, et alamsüsteemi sisendmuutujateks oleksid kas süsteemi primaarsed sisendid või registrid. Niisugune mudel võimaldab efektiivselt läbi viia nn. *tsüklipõhist simuleerimist,* kus iga süsteemi takti (tsükli) jaoks arvutatakse uus süsteemi olek registermuutujate uute väärtuste näol.

Üldjuhtumil võib registersiirete tasandil esitatud süsteemi käsitleda kui kompositsiooni juht- ja operatsioonautomaadist. Juhtautomaadi oleku- ja väljundmuutujaid võib seejuures interpreteerida vastavalt kui *mikrokäskude*

aadresse ja juhtsõnu (või juhtvälju) ja operatsioonautomaadi muutujaid kui andmesõnu.

Näide 1-18.

Vaatleme juhtautomaadist ja operatsioonautomaadist koosnevat digitaalsüsteemi joonisel 1-37.



Joonis 1-37. *Digitaalsüsteem*

Operatsioonautomaat koosneb kolmest registrist R_1, R_2 ja R_3, kolmest funktsionaalsest plokist - korrutajast, summaatorist ning spetsiaal-operatsioonelemendist *F*. Süsteemi andmesisendid on A ja B, ja andmeväljundiks on Y.

Juhtautomaat juhib andmeosa juhtsignaalidega y_1, y_2 ja y_3, ning juhtautomaadi sisendiks on operatsioonautomaadis genereeritav loogikatingimus $R_2 = 0$. Operatsioonautomaat on dekomponeeritud neljaks alamsüsteemiks väljundmuutujatega R_1, R_2, R_3 ja *C*. Alamsüsteemide funktsioonid on esitatud Tabelis 1-5.

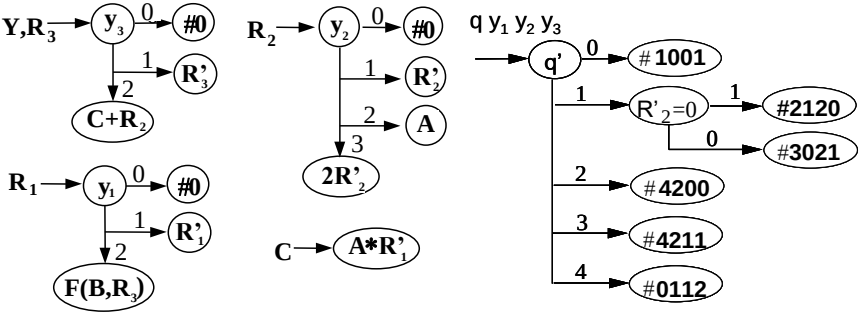
Ehkki tsüklipõhise modelleerimise idee kohaselt, kus alamsüsteeme moodustavad registrid koos oma sisendloogikatega, peaks korrutaja kuuluma registriga R_3 määratud alamsüsteemi, on siiski otstarbekam eraldada korrutaja ja summaator iseseisvate OD-mudeliga esindatud sõlmedeks. See võimaldab OD

terminaltippudega siduda iseseisvaid standardoperatsioone nagu $R_i * R_j$ ja $R_i + R_j$ ning vältida keerulisi ebastandardseid liitfunktsioone, näiteks $R_i * R_j + R_h$.

Tabel 1-5. *Digitaalsüsteemi alamsüsteemide funktsioonid*

Block	Control	RTL operation	Function
R_1	$y_1 = 0$	$R_1 = 0$	Reset
	$y_1 = 1$	$R_1 = R'_1$	Hold
	$y_1 = 2$	$R_1 = F(B, R'_1)$	Special
R_2	$y_2 = 0$	$R_2 = 0$	Reset
	$y_2 = 1$	$R_2 = R'_2$	Hold
	$y_2 = 2$	$R_2 = A$	Load
	$y_2 = 3$	$R_2 = 2 * R'_2$	Shift
R_3	$y_3 = 0$	$R_3 = 0$	Reset
	$y_3 = 1$	$R_3 = R'_3$	Hold
	$y_3 = 2$	$R_3 = C + R'_2$	Add
C	None	$C = A * R'_1$	Multiply

Joonisel 1-38 on esitatud OD-mudel digitaalsüsteemile joonisel 1-36. Mudel koosneb viiest otsustusdiagrammist $G_{q,y1,y2,y3,y4}$ juhtautomaadile ja graafid G_C, $G_{R,1}$, $G_{R,2}$, ja $G_{Y,R3}$ operatsioonautomaadi neljale alamsüsteemile.

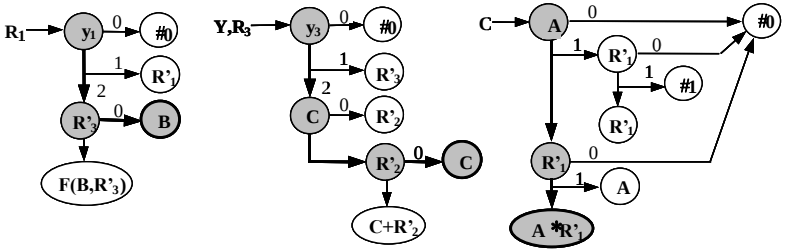


Joonis 1-38. *Otsustusdiagrammide mudel joonisel 1-32 esitatud süsteemile*

Tabel 1-6. *Juhtautomaadi üleminekute ja väljundite tabel*

State	Condition	Nstate	Control signals		
q’		q	y_1	y_2	y_3
0		1	0	0	1
1	$R_2 = 0$	2	1	2	0
1	$R_2 \neq 0$	3	0	2	1
2		4	2	0	0
3		4	2	1	1
4		0	1	1	2

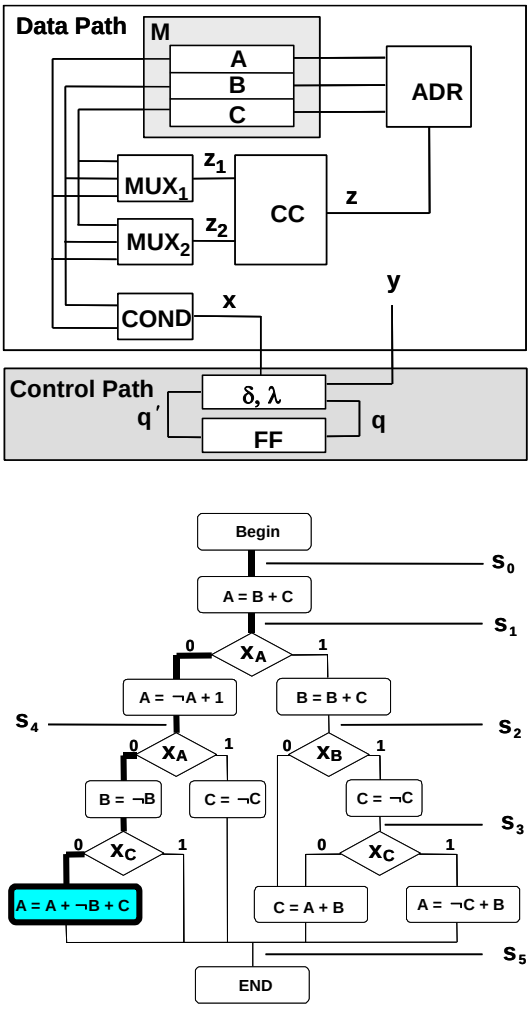
Juhtautomaadi üleminekute ja väljundite funktsioonid on esitatud Tabelis 1-6 ning otsustusdiagrammina $G_{q,y1,y2,y3,y4}$ joonisel 1-38. Mitteterminaalsed tipud on märgendatud eelmise oleku muutujaga *q*’ ja automaadi sisendmuutujatega x_1 ning x_2, terminaltipud aga – konstantidega, mis omistatakse graafi vektormuutujale *q*,*y*1,*y*2,*y*3,*y*4.



Joonis 1-39. *Transparentsusfunktsioonid otsustusdiagrammidel*

Joonisel 1-39 on demonstreeritud meetodit, kuidas saab kujutada OD-mudelis mitte üksnes alamskeemide funktsioone, vaid ka *transparentsuse* (“läbipaistvuse”) tingimusi, mis lubaksid läbi alamskeemide transportida stiimuleid ja veasignaale.

Näiteks, graafi G_{R1} on viidud lisaks 2 tippu R’$_3$ ja *B*, selleks et ilmutatult näidata, kuidas oleks võimalik juhtida vastavat alamskeemi, nii et see muutuks transparentseks signaalide saatmisel sisendist *B* registrisse R_1. Graafist on näha, et nimetatud töörezhiimi aktiveerimiseks tuleks tingimuste $y_1 = 2$ ja R’$_3 = 0$ abil aktiveerida tee algtipust y_1 terminaaltippu *B*.



Joonis 1-40. *Digitaalsüsteem ja tema mikroprogramm*

Graafis $G_{Y,R3}$, on esitatud töörezhiimid, mis võimaldavad aktiveerida transparentsed signaaliteed registrist R'_2 väljundisse Y (kui $C = 0$) registri R'_2 käitumise jälgimiseks ja korrutajast C väljundisse Y (kui $R'_2 = 0$) korrutaja käitumise jälgimiseks. Graafis $G_{Y,R3}$ on värvitult illustreeritud tee, mida tuleks aktiveerida, et väljundis Y saaks jälgida korrutaja väljundsignaale. Tee aktiveerimise tingimusteks on: $y_3 = 2$, $R'_2 = 0$. Summaator muutub nendel tingimustel transparentseks: 0 juurde liitmine korrutise tulemusele ei muuda selle väärtust.

Graafis G_C on lisatud neli täiendavat terminaaltippu, mis on tähistatud muutujatega R'_1 ja A ning konstantidega 0 ja 1. Uues graafis on esitatud korrutamisseadme töörezhiimid, mis võimaldavad aktiveerida transparentsed signaaliteed registrist R'_1 väljundisse C (kui $A = 1$) registri R'_1 käitumise jälgimiseks ja sisendist A väljundisse C (kui $R'_1 = 0$) sisendi A jälgimiseks. Lisaks on sisseviidud võimalus asetada muutuja olekutesse 0 või 1, aktiveerides tee terminaaltippu vastavalt valitud konstandiga. Värvitult on graafis $G_{Y,R3}$ illustreeritud tee, mida tuleks aktiveerida, et väljundis C saaks jälgida seadme põhifunktsiooni – korrutamist.

Kõigi nimetatud töörezhiimide (skeemi transparentseks tegemine, asetamine olekutesse 0 või 1) aktiveerimine on tüüpülesanded testide genereerimisel, mistõttu vajalike lahendite kiire leidmine mudelist on väga oluline.

Näide 1-19.

Joonisel 1-40 on esitatud digitaalsüsteem koos selle käitumist kirjeldava *mikroprogrammiga*. Süsteem koosneb nii juht- kui andmeosast.

Juhtosa on esitatud väljundfunktsiooniga $y = \lambda\,(q', x)$ ja siirdefunktsiooniga $q = \delta\,(q', x)$. Täisarvuline väljundmuutuja y esitab mikrokäsku - vektorit, mille komponentideks on mikrokäsuväljad $y = (y_M, y_z, y_{z,1}, y_{z,2})$, vastavalt mäluregistri valikuks *mikrooperatsiooni* (*mikrotehte*) tulemuse salvestamisel (y_M), mikrooperatsiooni valikuks (y_z,) ning mäluregistrite adresseerimiseks operandide valikul ($y_{z,1}$, $y_{z,2}$). Väljundfunktsiooni argumentideks on Boole'i sisendmuutujad $X = (x_A, x_C)$, mis esitavad andmeosa poolt genereeritavaid loogikatingimusi, ning täisarvuline olekumuutuja q', kus apostroof viitab olekule eelmisest taktist. Juhtautomaat on esitatav kahe otsustusdiagrammiga $q = G_q\,(q', x)$, $y = G_y\,(q', x)$,

Andmeosa koosseisu kuuluvad mäluplokk *M*, mis koosneb kolmest registrist *A, B, C* koos adresseerimisskeemiga *ADR*, mis on esitatavad kolme otsustusdiagrammiga $A = G_A\,(y_M, z)$, $B = G_B\,(y_M, z)$, $C = G_C\,(y_M, z)$,

andmetöötlusplokk *CC*, mida esitab graaf $z = G_z\,(y_z, z_1, z_2)$; ja kaks multiplekserit $z_1 = G_{z,1}\,(y_{z,1}, M)$ ning $z_2 = G_{z,2}\,(y_{z,2}, M)$. Loogikatingimuste arvutamiseks on plokk COND funktsiooniga $x = G_x\,(A, C)$.

Süsteemi kõrgtaseme struktuurne mudel on niisiis esitatav 9 otsustusdiagrammist koosneva süsteemina:

$$N_1=\{G_q, G_y, G_A, G_B, G_C, G_z, G_{z,1}, G_{z,2}, G_x\}.$$

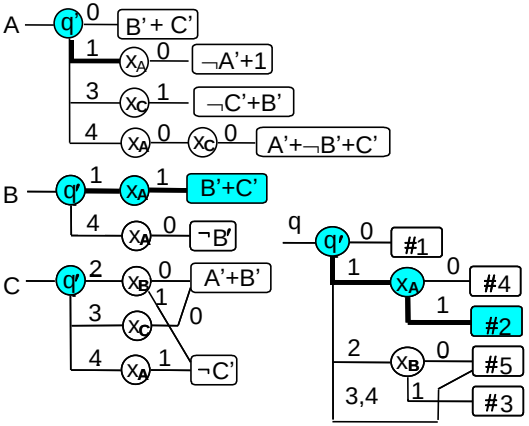
Superponeerides graafi $A = G_A\,(y_M, z)$ genereerime nüüd registri *A* modelleerimiseks uue kompaktsema mudeli:

$$A = G_A\,(y_M, z) = G_A\,(y_M, G_z\,(y_z, z_1, z_2)) =$$
$$= G_A\,(y_M, G_z\,(y_z, G_{z,1}\,(y_{z,1}, M), f_4\,(y_{z,2}, M))) =$$
$$= G_A\,(y_M, y_z, y_{z,1}, y_{z,2}, M) = G_A\,(y, M) =$$
$$= G_A\,(G_y\,(q', x), M) = G^*_A\,(q', A, B, C)$$

Korrates samasugust graafide superponeerimise võtet ka otsustusdiagrammidele G_q, G_B, G_C same kogu süsteemi esitamiseks uue kompaktsema mudeli:

$$N_2 = \{G_q, G^*_A, G^*_B, G^*_C\},$$

kus G_q esitab juhtautomaati ning graafid G^*_A, G^*_B, ja G^*_C, esitavad registreid *A*, *B* ja *C* koos sisendloogikaga. Süsteemi uus otsustusdiagrammide kompaktne mudel on esitatud joonisel 1-41.



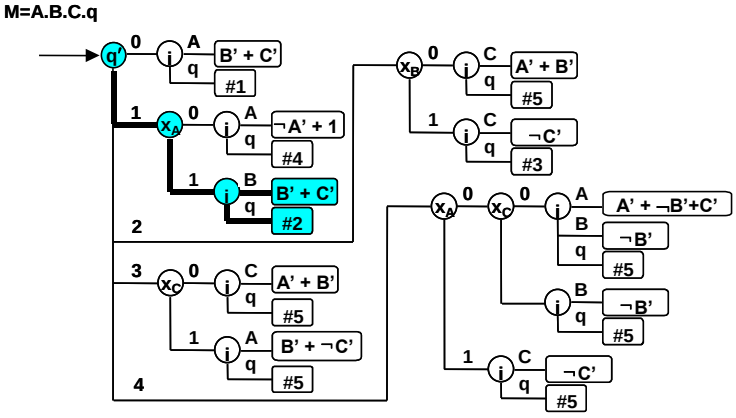
Joonis 1-41. *Joonisel 1-40 esitatud süsteemi otsustusdiagrammid*

Joonisel 1-42 on konstrueeritud sama digitaalsüsteemi esitav vektor-otsustusdiagramm $G_{q,A,B,C}$, kus kõik süsteemi esialgse mudeli graafid G_q, G_A, G_B, ning G_C on ühitatud ühte *vektorgraafi*. Graaf esitab *vektorfunktsiooni* $M = A.B.C.q$ ja võimaldab arvutada üheainsa sisenemisega graafi kogu süsteemi olekut ehk registrite *A,B,C* ja juhtautomaadi oleku *q* uusi väärtusi.

Vektorgraafide puhul on võimalik terminaaltippe esitada kahel viisil: kas esitada igale vektorikomponendile vastava avaldise eri tipuna nii nagu käesoleval juhul joonisel 1-42 või koondada vektoravaldised ühteainsasse tippu, nii nagu juhtautomaatide otsustusdiagrammides joonistel 1-29, 1-36, 1-38.

Esimesel juhul tuleb mudelisse viia uut tüüpi tipumuutuja – nn. *aadressimuutuja i*, mille ülesandeks on valida graafis haru vastavalt simuleeritavale vektori komponendile. *Aadresstipust*, mis on märgendatud aadressimuutujaga, väljub üldjuhul maksimaalselt *n* haru, kus *n* – on modelleeritava vektori komponentide arv. Juhul, kui mingi komponendi väärtus ei muutu simuleeritaval vektoril, siis vastav väljundharu aadresstipul puudub.

Nii näiteks, liikudes joonisel 1-42 piki märgistatud teed, sattume aadresstippu *i*, mis suunab meid arvutama vektori komponente *A* ja *q*. Nagu graafist näha, siirdudes olekust 0 olekusse 1, muutuja *M* komponendid *B* ja *C* säilitavad oma väärtusi, seega neid ei ole vaja adresseerida ega arvutada.



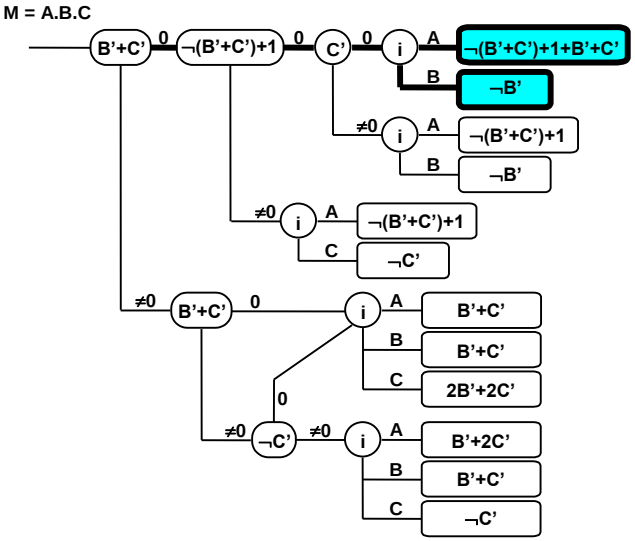
Joonis 1-42. *Joonisel 1-40 esitatud süsteemi vektorotsustusdiagramm*

Vektorotsustusdiagrammide kasutamise abil on veelgi võimalik tõsta süsteemi simuleerimise efektiivsust võrreldes komponentide võrgu simuleerimisega. Kui komponentide eraldi simuleerimisel eraldi graafe kasutades komponentmuutujate väärtuste arvutamine toimuks eraldi eri sammudena, siis vektorgraafis $G_{q,A,B,C}$ toimub süsteemi uue oleku arvutamine ühiselt kõigi funktsioonimuutujate *q*, *A*, *B* ja *C* jaoks korraga.

Joonisel 1-42 on halli värviga ja jämedate joontega illustreeritud võrreldav töömaht testvektori *q*'=1, x_A = 1 simuleerimisel. Vektorgraafil käiakse selle vektori simuleerimisel läbi 3 graafi sisetippu: *q*', x_A, ja *i*. Aadresstipul *i* on vaid kaks kaart, kuna muutujate *A* ja *C* väärtused antud testvektori korral ei muutu ja neid pole vaja arvutada. Sama testvektori simuleerimisel eraldi muutujate graafides (joonis 1-41) tuleks läbi käia 7 sisetippu 4-s graafis (4 korda *q*' ja 3 korda x_A), seega üle 2 korra rohkem.

Vektorotsustusdiagrammide komprimeerimine ajas

Digitaalsüsteemide simuleerimise kiiruse edasiseks tõstmiseks on võimalik teostada otsustusdiagrammides esitatud funktsioonide superponeerimist ka ajas.



Joonis 1-43. *Ajas superponeeritud vektorotsustusdiagramm*

Joonisel 1-43 on toodud digitaalsüsteemi esitus vektor-otsustusdiagrammina $G_{A,B,C}$, kus olekumuutujad ja ka olekugraaf G_q on elimineeritud ning kogu arvutusprotsess on taandatud ühele ainsale ajamomendile, millele vastab lõpphetk, kui mikroprogramm saab täidetud. Niisuguse vektorgraafi $G_{A,B,C}$ saamiseks, tuleks simuleerida takthaaval graafi $G_{q,A,B,C}$ üle kõigi olekumuutuja väärtuste q' = 0,1,2,3,4, ning fikseerida igal hetkel (vastava q' väärtuse juures) registermuutujate *A, B* ja *C sümbolväärtused* (avaldised nende muutujate arvutamiseks). Loogikatingimuste x_A ja x_C asemele graafi sisetippudes tuleb asetada nende muutujate arvutamiseks antud hetkel (antud q' väärtusel) kehtivad avaldised. Niisugust graafi manipuleerimist, mille tulemusena tekib uus graaf, nimetatakse *sümbolsimuleerimiseks*.

Sümbolsimuleerimise puhul ei asendata muutujaid nende konkreetsete väärtustega antud ajahetkel, vaid avaldistega, mille järgi neid tuleks hetkel arvutada. Seega otsustusdiagrammi sümbolsimuleerimise tulemusena ei teisendu esialgne graaf väärtuseks vaid uuesti graafiks (puuks).

Nii näiteks pärast vektormuutuja $M = A.B.C.q$ simuleerimist graafis $G_{q,A,B,C}$ (joonis 1-42) tingimusel q' = 0, saame tulemuseks $A = B' + C'$ ja q = 1. Vastav *harugraaf* alates kaarest q' = 0 joonisel 1-42 elimineeritakse. Edasi jätkatakse graafi simuleerimist väärtusel q' = 1. Loogikatingimus x_A (antud simuleerimise hetkel x_A on samane tingimusega $A = B' + C' \neq 0$ ja $x_A = 0$ on samane tingimusega $A = B' + C' = 0$) asendatakse graafis loogikatingimusega $B' + C' \neq 0$ ning jätkatakse graafi simuleerimist mõlema juhu jaoks: $B' + C' = 0$ ja $B' + C' \neq 0$.

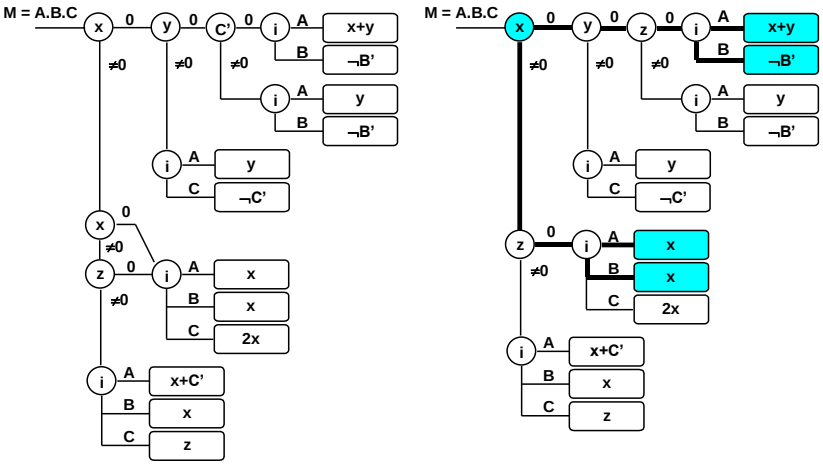
Jätkates nüüd sümbolsimuleerimist graafis $G_{q,A,B,C}$ mööda teed q' = 1, $x_A = 0$ (ehk $B' + C' = 0$), arvutame: $A = \neg A' + 1 = \neg(B' + C') + 1$ ja q' = 4. Vastav harugraaf graafis $G_{q,A,B,C}$ alates kaarest $x_A = 0$ elimineeritakse. Seejärel valime samas graafis uue haru q' = 4 ning siirdume tippu x_A. Loogikatingimus x_A (antud juhul x_A on samane tingimusega $A = \neg(B' + C') + 1 \neq 0$) asendatakse tingimusega $\neg(B' + C') + 1 \neq 0$ ning jätkatakse graafi simuleerimist mõlema juhu jaoks $\neg(B' + C') + 1 = 0$ ja $\neg(B' + C') + 1 \neq 0$.

Graafi $G_{q,A,B,C}$ (joonis 1-42) sümbolsimuleerimise tulemuseks on graaf $G_{A,B,C}$ (joonis 1-43), kus olekutipp q' on elimineeritud ja sisetippudes on Boole'i muutujad asendatud nende väärtuste arvutamiseks vajalike funktsionaalsete avaldistega.

Joonisel 1-44a on esitatud sama graaf, kus on kasutusele võetud abimuutujad vahepealsete tulemuste salvestamiseks:

$$x = B' + C',$$
$$y = \neg\,(B' + C') + 1 \qquad (1\text{-}35)$$
$$z = \neg C'.$$

Abimuutujate kasutamine võimaldab vältida korduvaid arvutusi. Joonisel 1-44b on veelkord esitatud sama otsustusdiagramm optimeeritud kujul, kus on elimineeritud muutuja *x*. Kerge veenduda, et mõlemad vektordiagrammid joonisel 1-44 on funktsionaalselt ekvivalentsed.



***Joonis 1-44**. Ajas superponeeritud abimuutujatega vektorotsustusdiagramm*

Võrreldes joonistel 1-43 ja 1-44a esitatud kompaktseid vektor-otsustusdiagramme mikroprogrammiga joonisel 1-40, võiks esimesel pilgul öelda, et erinevus peaaegu puudub. Nii on see tõepoolest traditsioonilist simuleerimist silmas pidades. Iga tee otsustusdiagrammides joonistel 1-43 ja 1-44 kujutab tegelikult ka mingit ühte teed läbi mikroprogrammi joonisel 1-40. Näiteks rasvaselt eristatud teele mikroprogrammis joonisel 1-40, mille tulemusena arvutatakse uued väärtused registrites *A* ja *B*, vastab joonisel 1-43 rasvaselt ära märgistatud harugraaf (ning, vastavalt, joonisel 1-44b rasvaselt märgistatud teed läbi sisetippude *x, y, z, i* terminaaltippudesse *x* + *y* ja ¬*B*').

Testide genereerimise aspekti silmas pidades on erinevus mikroprogrammi (joonis 1-40) ja otsustusdiagrammide vahel aga suur. Kui mikroprogramm modelleerib üksnes järjestikulist andmete arvutamise protsessi, siis otsustusdiagrammid esitavad selle kõrval veel ilmutatud kujul tulemus-põhjus vahekordi, mis võimaldab kergesti läbi viia diagnostilist analüüsi. Näiteks, avastades vigase tulemuse A^* registris A pärast mikroprogrammi läbimist jämedalt märgistatud teed mööda joonisel 1-40, oleks mikroprogrammi abil suhteliselt tülikas kindlaks teha võimalikku rikke mõjumispunkti (asukohta).

Joonisel 1-44b esitatud otsustusdiagrammis aga saaksime diagnostilist analüüsi suhteliselt lihtsalt korraldada ning vea asukoha lokaliseerida järgmisel viisil.

Vaadeldud operatsioonile (rasvaselt eristatud tee joonisel 1-40), mille tulemusena avastati viga A^*, vastab joonisel 1-44b aktiveeritud tee läbi tippude x, y, z. i ja $x + y$, kus muutujate x, y, z väärtused on arvutatud valemite (1-35) abil. Olgu operatsiooni tulemuseks saadud väärtus $A = A^* \neq x + y$. Läbitud tipud x, y, z, $x + y$ viitavad võimalikele vea põhjustele, millised tuleks üksteise järel läbi analüüsida.

Kui vea põhjuseks oleks sisetipp x, oleks graafis aktiveeritud tee sellest tipust alla ($x \neq 0$) läbi tippude z ($z = 0$), i ($i = A$), terminaaltippu x. Kui viga tõesti oleks sisetipus x, siis peaks operatsiooni tulemusena saadud vigase väärtuse jaoks kehtima ka tingimus $A^* = x$, kuna vea tõttu jõuti ju terminaaltipu $x + y$ asemel terminaaltippu x. Kui $A^* \neq x$, siis viga ei saa olla graafi sisetipus x ning tuleks analüüsida teisi läbitud tippe. Analoogiliselt eelnevale arutlusele leiame, et vea põhjuseks võiksid olla graafis läbitud sisetipud y või z, juhul, kui kehtiks $A^* = y$. Eristada neid veapõhjusi saaksime, kui analüüsiksime täiendavalt registri sisu C. Kui selle sisu oleks inverteeritud $C^* = \neg C'$, siis oleks vea põhjuseks tipp y, sest siis aktiveeruks graafis tee tipust y alla ($y \neq 0$) läbi tipu i ($i = C$), terminaaltippu $\neg C$. Vastasel juhul, kui registri C väärtus on jäänud muutumata $C^* = C'$, peab viga olema tipus z. Juhul kui $A^* \neq y$, võiks vea põhjuseks olla adresseerimistipp i, seda aga ainult juhul kui $A^* = \neg B'$. Kui ka see tingimus pole täidetud, $A^* \neq \neg B'$, siis saab vea põhjuseks olla vaid terminaaltipp $x + y$ ise.

Viimasel juhul on viga süsteemi operatsioonautomaadis (andmeosas), eelpoolsetel juhtudel oleks viga aga juhtautomaadis.

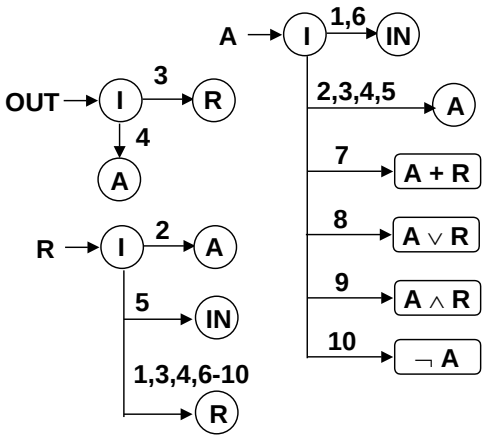
1.2.7. Otsustusdiagrammid mikroprotsessorite diagnostiliseks modelleerimiseks

Analoogiliselt registersiirete tasandil esitatud digitaalsüsteemidele on ka käsusüsteemidega kirjeldatud mikroprotsessorite diagnostiliseks modelleerimiseks võimalik kasutada otsustusdiagramme.

I_1: MVI A,D	A = IN	I_6: MOV A,M	A = IN
I_2: MOV R,A	R = A	I_7: ADD R	A = A + R
I_3: MOV M,R	OUT = R	I_8: ORA R	A = A $\vee$ R
I_4: MOV M,A	OUT = A	I_9: ANA R	A = A $\wedge$ R
I_5: MOV R,M	R = IN	I_{10}: CMA A,D	A = $\neg$ A

Joonis 1-45. *Hüpoteetilise mikroprotsessori käsusüsteem*

Otsustusdiagrammide sünteesi aluseks tuleks sel juhul võtta mikroprotsessorite käskude kirjeldused registersiirete tasandil. Käskude eristamiseks tuleks sisse viia uus registertaseme *käsumuutuja*, mille väärtusteks oleksid mikroprotsessori käsu numbrid või koodid.



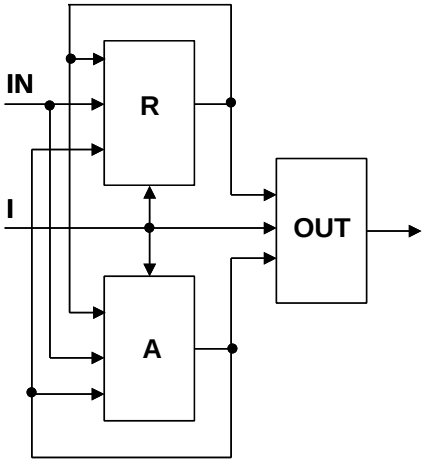
Joonis 1-46. *Mikroprotsessori käsusüsteemi otsustusdiagrammid*

Näide 1-20.

Joonisel 1-45 on toodud hüpoteetilise mikroprotsessori käsusüsteem ja sellele vastav otsustusdiagrammide võrk joonisel 1-46. Käsusüsteem sarnaneb tabelites 1-2 või 1-3 toodud töörezhiimide nimistuga ja otsustusdiagrammide süntees toimub analoogiliselt graafi G_{AC} sünteesile joonisel 1-24.

Joonisel 1-46 toodud graaf G_{OUT} vastab mikroprotsessori väljundkäitumisele ja graafid G_A ning G_R kirjeldavad vastavalt registrite *A* ja *R* sisendloogika funktsioone. Mudeli andmemuutujad on vektoriaalsed: sisendmuutuja *IN*, väljundmuutuja *OUT* ja registermuutuja *R*. Juhtmuutuja (käsumuutuja) *I*, mille väärtustele vastavad käsukoodid I_k, k = 1,2,...,10, on täisarvuline.

Otsustusdiagrammide mudeli erinevuseks võrreldes käsusüsteemiga on see, et põhitähelepanu on pööratud mitte sellele, mis toimub ühe või teise konkreetse käsu ajal, vaid sellele, kuidas käituvad eri käskude puhul mikroprotsessori käsusüsteemi sise- ja väljundmuutujad.



Joonis 1-47. *Mikroprotsessori struktuurskeem*

On lihtne märgata, et graafid OD-mudelis on muutujate kaudu omavahel seotud. Näiteks graafimuutujad *R* ja *A* on argumentideks ehk muutujateks väljundgraafis G_{OUT}. Seega sisuliselt kujutab graafide süsteem

joonisel 1-46 endast võrku, mis on esitatav mikroprotsessori käitumist kujutava struktuurskeemina joonisel 1-47.

Teiste sõnadega, minnes traditsiooniliselt mikroprotsessori *protseduurselt mudelilt* üle otsustusdiagrammide mudelile saame uue struktuurse esituse, mis võimaldab vahetumalt formuleerida testide sünteesi ja analüüsi ülesandeid, kasutades traditsioonilisi struktuurseid testide sünteesi ja analüüsi meetodeid. Digitaalsüsteemide diagnostika otsesteks objektideks on rikked, aga rikkeid saab vahetumalt modelleerida *struktuursetel* mudelitel võrreldes protseduursete või *funktsionaalsete* (musta kasti tüüpi) esitustega.

Detailsemaks mikroprotsessori kirjeldamiseks *käsusüsteemi tasandil* oleksid järgmised võimalused:

1) Mikroprotsessori *käsuformaadi* võib dekomponeerida väljadeks, kus igal väljal on oma spetsiifiline funktsioon: *operatsioonikood,* esimese registri aadress, teise registri aadress jne. Igale väljale võib vastavusse seada eri muutuja, mida võib vaadelda käsumuutuja kui vektormuutuja ühe komponendina.

2) Käsutäitmise protsessi võib vaadelda *käsutsüklitena* ja niisuguste tsüklite esitamiseks võib kasutusele võtta analoogiliselt automaatide olekumuutujale *tsüklimuutuja*. Tsüklimuutuja väärtustele vastaksid käsu täitmise erinevad tsüklid: käsu väljalugemine, esimese operandi lugemine, teise operandi lugemine, tulemuse kirjutamine mällu jne. Juhul, kui tsüklite arv on konstantne iga konkreetse käsu jaoks ja üleminek olekust olekusse ei sõltu andmetest, võib olekumutuja graafist loobuda ning käsitleda olekumuutujat analoogiliselt aadressmuutujale vektorgraafides, viies aadressdimensiooni kõrval mudelisse sisse ka aja dimensiooni.

1.3. Kokkuvõtteks

Käesolevas peatükis vaadeldi kahte digitaalsüsteemide diagnostilise modelleerimise matemaatilist aparaati: Boole differentsiaalalgebrat ja graafiteooriat ehk otsustusdiagramme.

Boole differentsiaalaparaadi tähendus praktiliste diagnostikaülesannete keerukust silmas pidades on rohkem tunnetuslik. Konkreetselt lahendada saab selle aparaadi abil vaid lihtsaid testide genereerimise, rikete simuleerimise ja diagnostika ülesandeid mittekeerukate loogikaskeemide jaoks. Mikroprotsessori puhul pole selle aparaadiga kahjuks midagi teha.

Küll on aga Boole'i differentsiaalaparaat oma kompaktsuse ja korrektsuse poolest sobiv testide sünteesi ja analüüsi matemaatilise probleemi formuleerimiseks ning probleemide vaheliste seoste lihtsaks kirjeldamiseks.

Suurepäraseks näiteks oleks siin testide genereerimise ja rikete diagnostika probleemide ühtne formuleerimine diagnostika üldvõrrandi abil (alapunktis 1.1.8), mis võimaldab mõlemat probleemi vaadelda teineteise pöördprobleemina.

Teiseks iseloomulikuks näiteks on olemusliku selgituse andmine traditsioonilisele insenerlikule lihtsustusele, kus piisavaks loetakse testide genereerimist üksikutele riketele, ignoreerides nende kordse esinemise võimalust. Selgituse sellele probleemile saame alapunktis 1.1.7, kus näeme, kuidas Boole'i osatuletis, mille abil formuleeritakse testide genereerimise ülesannet, tuleneb erijuhuna Boole'i täisdifferentsiaalist, mis modelleerib skeemi kõikide võimalike rikete ja nende kombinatsioonide korral.

Samas omab Boole'i differentsiaalaparaat ka praktilist tähendust füüsikaliste defektide loogilisel modelleerimisel, mida käsitletakse järgmises peatükis.

Niisiis, Boole'i differentsiaalaparaat võimaldab korrektselt formuleerida probleemi. Kui probleem on formuleeritud, saab lahendamiseks otsida juba sobivamaid meetodeid ja mudeleid. Ühe niisuguse mudelina sai käesolevas raamatus valitud otsustusdiagrammide mudel.

Kui enamus digitaalsüsteemide diagnostika probleeme loogikaskeemide tasandil on peaaegu täiuslikult lahendatud Boole'i algebrat ja Boole'i differentsiaalalgebrat kasutades, siis kõrgemate tasandite jaoks vastav matemaatiline aparatuur on kaua puudunud. Samas on keerukate miljonitest loogikaelementidest koosnevate süsteemide diagnostikaprobleemide lahendamine ainuvõimalik kõrgematel tasanditel "töötavate" meetodite kaasamisega. Niisugusteks tasanditeks oleksid protseduurse register-taseme kirjeldused (näiteks VHDL keeles), protsessorite käsusüsteemid jne.

Kui Boole'i algebral põhinevaid diagnostikameetodeid ei ole võimalik otseselt üldistada kõrgematele tasanditele, siis graafiteoorial põhinevat topoloogilist käsitlust kasutades osutub üldistamine võimalikuks.

Üldjoontes seisneb niisuguse üldistamise võimalus järgnevas. Kõigepealt "tõlgime" Boole'i algebra meetodid ümber topoloogilisteks meetoditeks, kasutades Boole'i algebra ühte tööriista binaarseid otsustusdiagramme. Seejärel eemaldame otsustusdiagrammidelt binaarsuse kitsenduse ja rakendame juba olemasolevaid "ümbertõlgitud" topoloogilisi meetodeid

neid küll pisut modifitseerides juba kõrgematel tasanditel kirjeldatud süsteemide puhul.

Käesolevas peatükis vaadeldud meetodeid kasutame nüüd edaspidi konkreetsete insenerlike diagnostikaprobleemide käsitlemisel. Järgmises peatükis vaadeldakse diagnostika alusprobleemi – rikete modelleerimist. Edaspidi aga võtame käsile otsesed diagnostikaprobleemid, millisteks on: testide genereerimine, rikete simuleerimine ja rikete diagnostika.

2. Rikete modelleerimine digitaalsüsteemides

Füüsikaliste *defektide* põhjusi on väga palju ja võimalike defektide arv on väga suur. Pole mõeldav defektide ükshaaval käsitlemine ei testide genereerimisel ega ka defektide diagnoosil. Samal ajal aga tuleks *testimise kvaliteeti* mõõta avastatavate defektide kattega ehk protsendiga kõigi võimalike defektide suhtes. Niisugune kvaliteedi mõõtmine on praktiliselt võimatu. Seetõttu tuleks defekte modelleerida ja testimise kvaliteeti mõõta füüsikalise tasandiga või transistorskeemidega võrreldes kõrgematel digitaalsüsteemide abstraktsetel esitustasanditel.

Rikete modelleerimise eesmärgiks on asendada praktiliselt lõpmatu võimalike defektide hulk, sellega samaväärse lõpliku rikete hulgaga. *Rikete mudeleid* kasutatakse nii testide genereerimisel kui ka füüsikaliste defektide lokaliseerimiseks ning neid tuleks vaadelda kui silda, mis ületab lõhet füüsikalise reaalsuse ja abstraktse matemaatika vahel.

Kõige sagedamini on kasutatud ja kasutatakse *konstantse rikke mudelit*, mille järgi digitaalskeeme vaadeldakse kui ventiilide võrku ja võimalikeks riketeks on loogikakonstandid 1 või 0 ventiilide sisendites ja väljundites. Mitmete selle rikkemudeli omaduste ära kasutamine aitab vähendada otsest käsitlemist nõudvate konstantsete rikete hulka.

Samal ajal on selgeks saanud, et konstantsete loogikarikete mudel on siiski piiratud. Ühest küljest kasvab loogikataseme käsitlemist vajavate rikete hulk liiga kiiresti süsteemide keerukuse kasvades. Teiselt poolt aga ei kata see rikkemudel piisavalt adekvaatselt *transistorskeemide* reaalseid defekte. Käesoleva peatüki eesmärgiks on: uurida, kuidas saaks käsitleda rikkeid ja tagada kõrget testimise kvaliteeti üha keerukamaks muutuvates digitaalsüsteemides. Vaatleme, kuidas on võimalik kasutada Boole'i differentsiaalaparaati füüsikaliste defektide kujutamiseks loogikatasandile ja

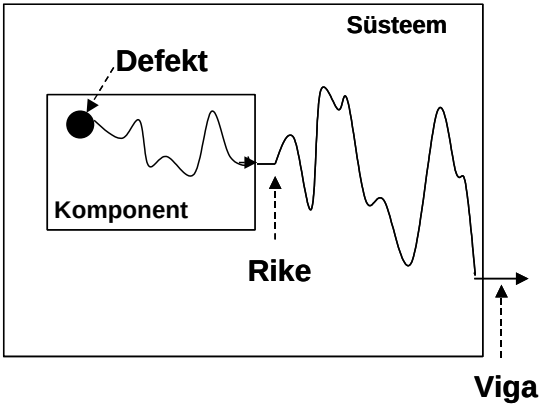
milline peaks olema rikete mudel, mis poleks väga keerukas, aga samas ometi esindaks täpsemalt reaalset füüsikalist defektide maailma.

2.1. Rikete klassifikatsioon

Alljärgnevas peatükis vaatleme, mille poolest erinevad üksteisest niisugused mõisted nagu defektid, rikked ja vead. Uurime põhilisi rikete tüüpe, nende mudeleid ning rikete põhiomadusi.

2.1.1. Defektid, rikked, vead

Terminoloogia korrastamiseks püüame kõigepealt defineerida mõningad põhimõisted. Termineid *defekt, rike* ja *viga* kasutatakse nii teaduslikus kirjanduses kui ka elus sageli erinevates ja kattuvates tähendustes. Aga need mõisted ei ole sünonüümid. Käesolevas õppevahendis kasutame definitsioone, mis on esitatud defektide ja nende testimisele pühendatud monograafias [Ben 98].



Joonis 2-1. *Defekt, rike ja viga digitaalsüsteemis*

Defektiks elektroonikasüsteemis nimetame mittekavandatud erinevust disaini ja selle implementatsiooni vahel riistvaras. Defekt kutsub esile

erinevuse elektroonskeemis võrreldes selle spetsifikatsioniga. Defekt tekib kas seadme valmistamisprotsessis või tema kasutamise ajal.

Rikkeks nimetame defekti kujutamist abstraktsel funktsionaalsel tasandil. Rike on defekti matemaatiline mudel. Joonisel 2-1 on kujutatud defekt, rike ja viga omavahelises suhtes. Kusagil transistorskeemi tasandil tekib *füüsikaline defekt*, mis avaldub selles, et defektse komponendi väljundsignaal võtab väära püsiva väärtuse, mida võib tõlgendada loogikatasandil kui konstantset väära loogikaväärtust ehk *riket*. Seda väärtust kasutatakse skeemi loogikamudeliga manipuleerides, näiteks testi genereerides, mille ülesandeks on antud defekti avastamine ehk rikke „levitamine" väljundisse, kus signaalid oleksid jälgitavad. Väljundisse levitatud väär signaal on tõlgentatav avastatud *veana*.

Veaks nimetamegi väärat väljundsignaali, mida põhjustab defektne süsteem. Vea põhjuseks on mingi defekt.

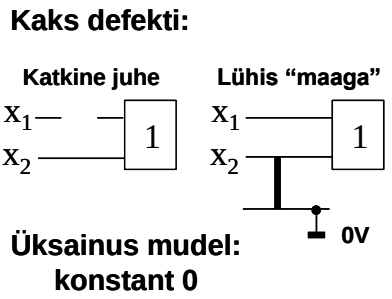
2.1.2. Rikete mudelite üldised omadused

Füüsikaliste *defektide modelleerimisel* abstraktsete rikete mudelite abil on terve rida olulisi motivatsioone:

- väheneb defektide käsitlemise keerukus: mitmeid erinevaid füüsikalisi defekte esindab üksainus rikke mudel (joonis 2-2);
- ühtainust matemaatilist rikke mudelit võib kasutada erinevate tehnoloogiate puhul, kus defektide füüsikalised tähendused võivad erineda;
- rikete loogikamudeleid võib edukalt kasutada juhtumitel, kui rikete füüsikaline tähendus ja effekt pole päris selge;
- rikete mudelid annavad võimaluse töötada füüsikalisest tasandist kõrgematel abstraktsetel tasanditel.

Näide 2.1.

Joonisel 2-2 on näidatud kaks defekti: katkine juhe ja lühis maaga. Mõlemad defektid on loogiliselt modelleeritavad ühe ja sama rikkena – konstantne 0.



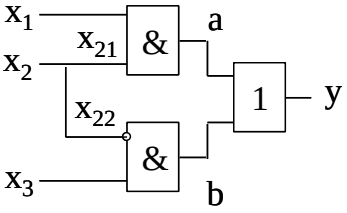
Joonis 2-2. *Kahte defekti modelleerib üksainus rike*

Rikete mudelid võivad olla *otsesed* ja *kaudsed*.

Otsese rikkemudeli puhul võib rikkeid loendada. Näiteks konstantne loogikarike on otsene, sest ta on loendatav. Kui skeemis on elementide vahel *n* ühendust ja igal ühendusel võib olla kaks riket – konstantne 0 ja konstantne 1, siis kõigi rikete arv skeemis on 2*n*. *Kaudsed rikked* ei ole loendatavad, nad defineeritakse harilikult mingi iseloomuliku omadusena.

Rikete mudelid võivad olla *struktuursed* ja *funktsionaalsed*.

Struktuursed rikked on seotud konkreetse skeemi struktuuriga, nad näiteks modifitseerivad komponentide vahelisi ühendusi. *Funktsionaalsed rikked* aga on seotud komponentide funktsionaalsete mudelitega, nad näiteks modifitseerivad komponentide funktsioone.



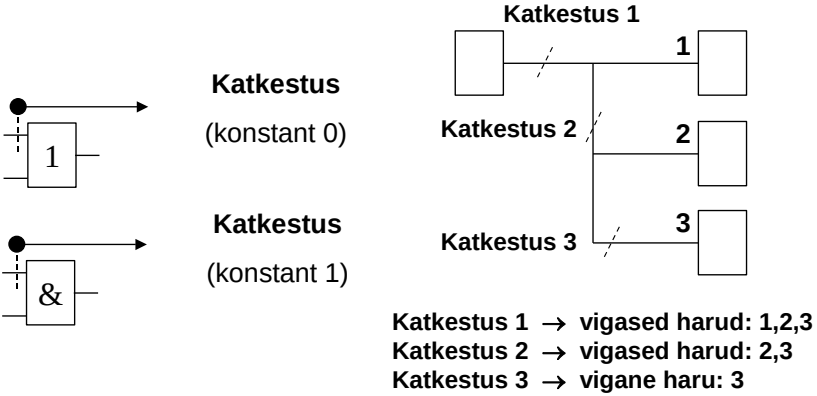
Joonis 2-3. *Kombinatsioonskeem*

Näide 2-2.

Vaatleme skeemi joonisel 2-3. Struktuurseteks riketeks selles skeemis võivad näiteks olla: katkine juhe *a*, lühis juhtmete x_1 ja x_2 vahel või vigane komponent. Funktsionaalse rikke näiteks on aga funktsiooni muutus: õige funktsiooni asemel realiseerub funktsioon:

$$y = x_1 x_2 \vee \overline{x_2} x_3 \Rightarrow y = \overline{x_2} x_3 .$$

Struktuursete rikete puhul eeldatakse vahel ka, et skeemi komponendid on korras ja rikked võivad olla ainult komponentide vahelistes ühendustes.



Joonis 2-4. *Mittevastavus defektide ja konstantse rikke mudeli vahel*

Mitte alati pole ventiilskeemides võimalik luua adekvaatset vahekorda reaalsete struktuursete rikete ja konstantse rikke mudeli vahel. Näiteks katkine juhe võib tähendada erinevate loogikaelementide puhul nii konstanti 0 kui ka konstanti 1 (joonis 2-4). Kui juhe hargneb, siis olgugi, et on tegemist ühe ja sama muutujaga tuleb ometigi katkestusi erinevates kohtades modelleerida erinevalt (joonis 2-4).

2.1.3. Konstantsed rikked

Konstantseid rikkeid (*SAF mudel – stuck-at fault*) modelleeritakse fikseerides konstantse väärtuse signaalijuhtmele, millele korras skeemis vastaks Boole'i muutuja. Signaalijuhtmed on loogikaskeemides ventiilide või trigerite sisenditeks või väljunditeks. Eeldatakse, et konstantsed rikked mõjutavad vaid ventiilide vahelisi ühendusi, aga mitte ventiile endid. Iga ühendusjuhe võib omada kahte tüüpi rikkeid: *konstant* 0 (*stuck-at-0, SAF0*) või *konstant* 1 (*stuck-at-1, SAF1*). Konstantse rikke näited on joonisel 2-4.

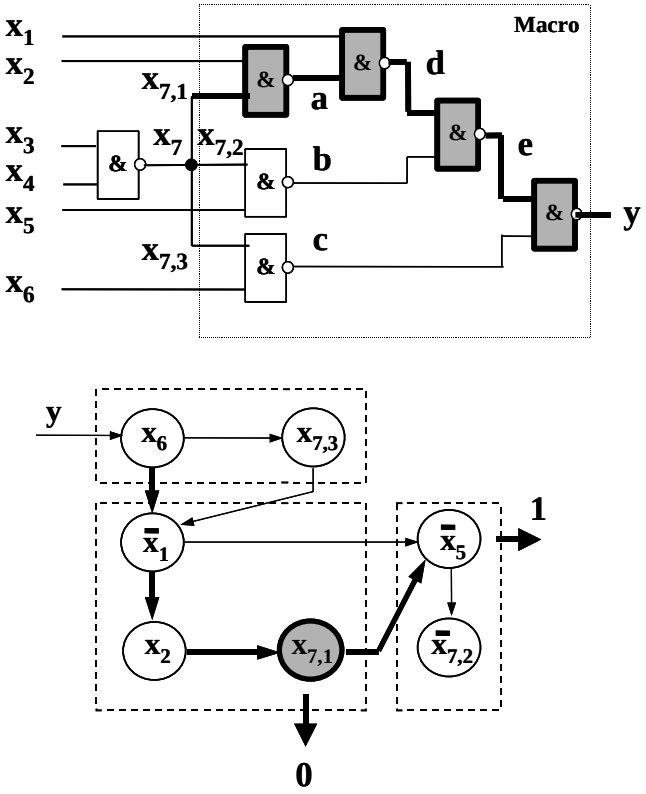
Üksiku konstantse rikke mudel on kõige levinenum rikke mudel digitaalskeemide ja süsteemide diagnostikas: igal ajahetkel eeldatakse vaid ühe rikke olemasolu, kusjuures see rike on püsiv (*permanentne*)[3]. Üksiku konstantse rikke mudelil on järgmised omadused:

- ainult üks signaalijuhe võib olla vigane;
- signaalijuhtmel on püsivalt konstantne signaal 1 või 0;
- konstantne signaal võib olla kas ventiili sisendis või väljundis.

Konstantse rikke mudel sai tööstuslikuks standardiks 1959. aastal. On ennustatud konstantse rikke mudelist loobumist, kuid mitmetel põhjustel ja mitmete konstantse rikke omaduste tõttu on ta endiselt laialt kasutusel. Niisugusteks omadusteks on:

- mudeli lihtsus;
- selle käsitlemise hõlpsus;
- mudeli loogiline käitumine – simuleerimine on vahetu ja deterministlik;
- mõõdetavus: avastamise või mitteavastamise kindlaks tegemine on lihtne;
- kohandatavus: mudelit saab kergesti rakendada suvalistel süsteemi abstraktse esituse tasanditel: ventiilid, transistorid, süsteemid, registersiirete tasand jne.

[3] Eksisteerib ka *ajutise (transient) rikke* mudel, mida käesolevas õppevahendis ei vaadelda



Joonis 2-5. *Testi genereerimine konstantsele rikkele*

Näide 2-3.

Vaatleme testi genereerimist konstantsele rikkele $x_{71} \equiv 0$ skeemis joonisel 2-5. Kasutades *Boole'i osatuletisi*, tuleks lahendada differentsiaalvõrrand

$$\frac{\partial y}{\partial x_{71}} = \frac{\partial(x_6 x_{73} \vee (\overline{x_1} \vee x_2 x_{71})(\overline{x_5} \vee \overline{x_{72}}))}{\partial x_{71}} = 1$$

lisatingimusel

$$x_7 = 1.$$

Kasutades SSBOD mudelit joonisel 2-5 tuleks

- aktiveerida tee tipuni x_{71}, omistades teel kohatud muutujatele väärtused: $x_6 = 0$, $x_1 = 1$, $x_2 = 1$,
- fikseerida muutuja x_7 väärtus $x_7 = 1$,
- ning aktiveerida tipust x_{71} tee terminali 1, fikseerides $x_5 = 0$.

Väärtuse $x_7 = 1$ tagamiseks sisenditel tuleks lahendada võrrand $\overline{x_3 x_4} = 1.$

Vaatamata konstantse rikke mudeli väga headele omadustele, ei saa nii mitmeidki reaalseid füüsikalisi defekte ometi modelleerida tema abil, põhiliselt just *CMOS tehnoloogia* puhul. See on konstantse rikke mudeli puuduseks. Mõnikord püütakse mudeli "võimsust" suurendada lubades *kordsete rikete* olemasolu. Kuid eriti see siiski ei suurenda reaalsete defektide katet, pealegi on kordse rikke mudel väga keerukas: rikete arv n signaalijuhtmega skeemis on 3^n. Testide genereerimise ja rikete simuleerimise algoritmid muutuvad kordsete rikete mudeli puhul samuti väga keerukaks.

2.1.4. Lühised, katkestused ja viiterikked

Tulenevalt konstantse rikke puudustest on võetud kasutusele terve rida täiendavaid rikete mudeleid nagu lühised, katkestused, viiterikked jne. Teatud rikkeid tuleks modelleerida ainult transistoride tasemel. Rikete modelleerimise keerukuse vähendamiseks on püütud välja töötada ka palju kõrgtaseme rikkemudeleid.

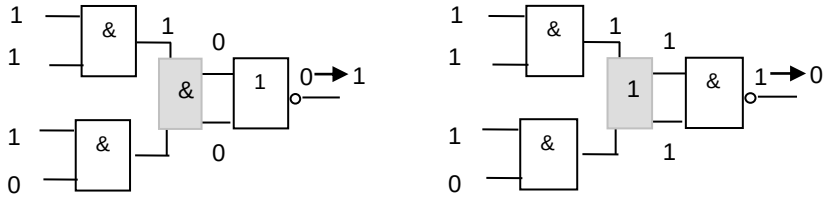
Lühised.

Lühised (*bridges, shorts*) modelleerivad kõiki neid füüsikalisi defekte, mis põhjustavad mittekavatsetud elektrilisi kontakte kahe või rohkema signaalijuhtme vahel.

Lühiseid võib iseloomustada lineaarsete või mittelineaarsete omadustega, lühistakistuse omades väärtusi vahemikus 0 kuni 1 MΩ.

Teatud kriitilistest takistuse vahemikest väljaspool reaalne lühis võib ka mitte enam põhjustada rikke effekti.

Lühised võib jämedalt jagada kahte gruppi: *ventiilide vahelised* lühised (*inter-gate shorts*), *ventiilisisesed* ehk transistoride lühised (*intra-gate shorts*).



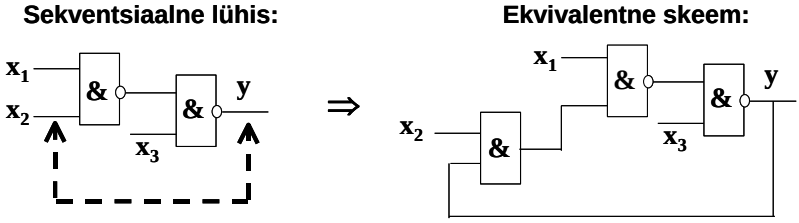
Joonis 2-6. *Ventiilide vahelised kombinatoorsed lühised*

Ventiilide vahelisi lühiseid võib omakorda klassifitseerida järgnevalt:

- JA-tüüpi lühised (wired-AND short) või 0-domineerimisega lühised (vt. näidet joonisel 2-6a);
- VÕI-tüüpi lühised (wired-OR short) või 1-domineerimisega lühised (vt. näidet joonisel 2-6b);
- Mitte määratud tüüp, kus lühis sõltub tehnoloogiast, mida kasutatakse.

Lühised võivad olla *kombinatoorsed* ja *sekventsiaalsed*.

Sekventsiaalsed lühised põhjustavad kombinatsioonskeemides sekventsiaalset käitumist, siis kui lühis tekitab elektrilise tagasiside (joonis 2-7).

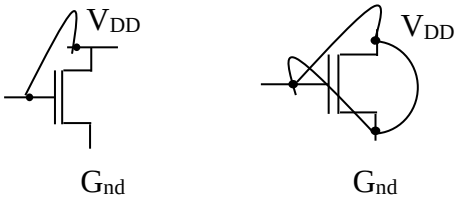


Joonis 2-7. *Ventiilide vaheline sekventsiaalne lühis*

Kombinatoorsete lühiste testimise protsent konstantse rikke mudelit kasutades on harilikult kõrge. Sekventsiaalsete lühiste testimine on aga väga keerukas, sest testide genereerimise meetodid üldiselt eeldavad, et olekute arv skeemis ei kasva (sekventsiaalne lühis põhjustab aga täiendava oleku tekkimise skeemis) ning et kombinatsiooniskeemid rikke puhul jäävad ikka kombinatsioonskeemideks (sekventsiaalse rikke tõttu muutub aga kombinatsioonskeem sekventsiaalseks ehk järjestikskeemiks).

Lühised transistorskeemides

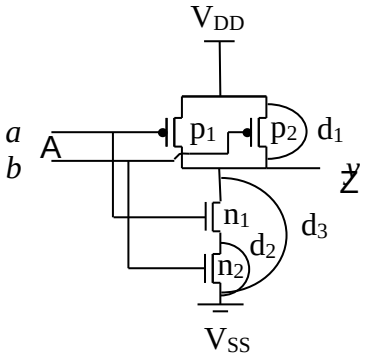
Transistorskeemides võib põhimõtteliselt eristada 6 tüüpi lühist kõikide transistori terminalide vahel: ventiil, *neel* (*drain*), *läte* (*source*), *põhimik* (*substrate*) ja lühised V_{DD} ja G_{nd} sõlmedega (joonis 2-8). Mainitud rikete tegelik avaldumine sõltub lühistakistuse suurusest. Mõningaid ventiilisiseseid transistortaseme rikkeid võib modelleerida ka konstantse rikke mudeliga.



Joonis 2-8. *Transistori lühised*

Näide 2-4.

Joonisel 2-5 on illustreeritud *CMOS tehnoloogias* realiseeritud JA-EI ventiili transistorskeem, milles on kolm lühisriket d_1 , d_2 , d_3.



Joonis 2-9. *Lühised transistortaseme CMOS JA-EI ventiilis*

Eeldades, et lühistakistus on oluliselt madalam kui avatud transistori sisetakistus, võib defekte d_1 and d_3 modelleerida loogikatasandi konstantse rikke mudeliga, aga defekti d_2 mitte.

Defekt d_1 põhjustab lühise toiteallika VDD ja väljundsõlme y vahel. Rakendades testvektorit $(a,b) = 11$, avanevad transistorid n_1 ja n_2. Eeldades, et avatud transistoride sisetakistus on oluliselt kõrgem defekti d_1 põhjustatud lühistakistusest, võib defekti lugeda ekvivalentseks konstant 1 rikkega väljundis y.

Analoogilise põhjenduse saab anda ka defekti d_3 taandamisele konstantseks rikeks $d_3 \equiv 1$.

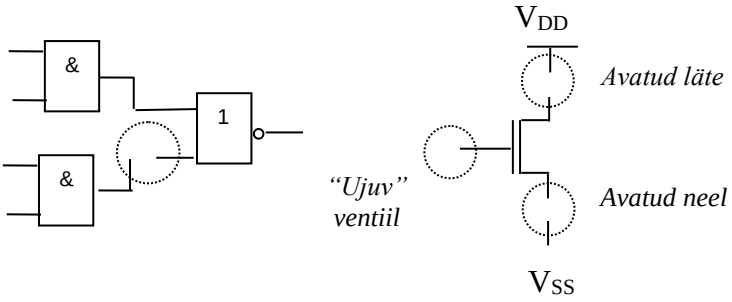
Defekti d_2 võib avastada konstantse rikke testiga eeldusel, et lühisrikke takistus ja avatud transistori n_1 takistus on oluliselt väiksemad kui avatud transistori p_2 takistus. Sellise tingimuse täitmisel avastab defekti d_2 testvektor $(a,b) = 10$, mis avab transistorid n_1 ja p_2.

Katkestused

Katkestusteks (*opens*) nimetatakse järgmist tüüpi defekte:

- kontakti puudumine ühendustes,
- katkenud juhtmed,
- praod metalliseeritud kihtides jne.

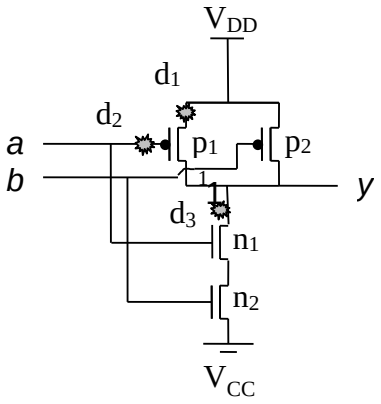
Katkestuste käitumine varieerub väga palju ja neid on tihti raske ette ennustada. Katkestuste avaldumine ei sõltu üksnes defekti suurusest või ulatusest vaid ka temperatuurist, taktsagedusest, asukohast, tehnoloogilistest parameetritest jne. Katkestused võivad lokaliseeruda nii ventiilide tasandil kui ka transistoride tasandil (joonis 2-10).



Joonis 2-10. *Katkestused ventiilskeemis ja transistoris*

Näide 2-5.

Joonisel 2-11 on illustreeritud CMOS tehnoloogias realiseeritud JA-EI ventiili transistorskeem, milles on kolm katkestusdefekti d_1, d_2, d_3. avaldades mõju transistori p_1 käitumisele.



Joonis 2-11. *Katkestused* CMOS JA-EI transistorskeemis

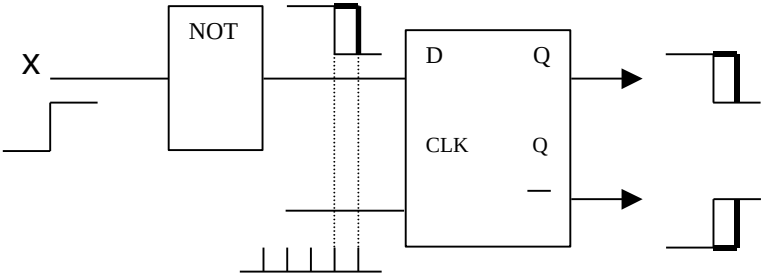
Konstantse rikke mudelit kasutades genereeritakse antud JA-EI elemendile järgmine testvektorite (a,b) hulk: T_1 = (11), T_2 = (10), T_3 = (01). Vaadeldavad katkestusdefektid aga ei ole avastatavad ühegi üksiku testvektoriga sellest hulgast. Vektor T_2 viib väljundi väärtusele 1 transistori p_2 abil ja vektor T_3 viib väljundi väärtusele 1 transistori p_1 abil. Defektide d_1, d_2, d_3 olemasolul aga vektor T_3 ei vii transistori olekusse 1. Siiski, nimetatud testvektorite kolmik avastaks need defektid, kui vektorid viidaks läbi niisuguses järjestuses: T_2, T_1, T_3.

Üldjuhtumil on katkestusdefekti avastamiseks transistorskeemides vaja kahe testvektori jada: T_1, T_2. Esimene vektor T_1 initsialiseerib skeemi väljundi vajalikule loogikatasemele, teine vektor T_2 üritab muuta skeemi olekut testitava transistori abil.

Viiterikked

Ajastus (*timing*) ehk ajalised *viited* (*delay*) on väga olulised parameetrid sisend-väljundsignaalide suhetes. Defektide omaduste uurimine on näidanud, et enamik CMOS transistoride defekte põhjustavad just viiterikkeid (viiteid signaalide levikus), võrreldes näiteks teiste nn. *katastroofiliste* (püsivate) defektidega. Näiteks nn. *takistuslikud* lühisrikked (*resistive* bridges) lühistakistusega väljaspool kriitilise takistuse alasid põhjustavad just viiteid signaalide levikus e. viiterikkeid. Viiterikked kuuluvad defektide klassi, mis ei kuulu ei lühiste ega ka katkestuste

kategooriasse. Viiterike tähendab seda, et signaal hilineb väljundisse ja seetõttu konkreetsel hetkel (takti lõpus) oodatav väljundsignaal on vigane (joonis 2-12).



Joonis 2-12. *Viiterike*

Viiterikked võivad tekkida paljudest põhjustest tingitult, mis võiksid olla, näiteks:

- tehnoloogilise protsessi defektid;
- transistoride *lävipingete* nihked;
- *parasiitmahtuvuste* suurenemine;
- ebatäpne signaalide ajastuse projekteerimine.

Viiterikkeid võib seostada nii ventiilidega kui ka signaaliteedega, mistõttu viiterikete mudeleid võib klassifitseerida järgmiselt:

- *ventiili viiterikke* mudel (digitaalskeeme projekteeritakse lähtudes elementide nominaalsetest viidetest; defekt ventiilis aga võib põhjustada nominaalsest palju suurema viite);
- *signaalitee viiterikke* mudel (mudel võtab kumulatiivselt arvesse kõigi ventiilide viiterikked antud signaaliteel)

2.2. Konstantsete rikete omadused

Üks olulisemaid probleeme testvektorite genereerimisel ja rikete simuleerimisel digitaalsüsteemides on skeemide keerukus – hiiglasuur rikete hulk, kus iga rikke kohta tuleb mingi seisukoht võtta (genereerida talle test või kindlaks teha, kas konkreetne olemasolev test suudab seda riket avastada või mitte).

Probleemist üle saamiseks tuleks mingil moel püüda seda rikete hulka vähendada, määrata kindlaks antud rikke olulisus, suhe teistesse riketesse. Et niisugust analüüsi läbi viia, peaks tundma erinevaid rikete omadusi nagu testitavus, liiasus, ekvivalentsus teiste riketega, dominantsus teiste rikete suhtes jne. Tundes rikete niisugusi omadusi, osutub võimalikuks selekteerida välja kogu rikete hulgast mingi olulisemate rikete alamhulk ja tegelda edaspidi vaid nende välja valitud riketega, teades sel juhul näiteks, et kui testid on võimelised avastama kõik rikked sellest valikhulgast, avastavad nad ka kõik ülejäänud rikked.

2.2.1. Rikete liiasus

Olgu $y = f(X)$ mingi kombinatsioonskeemi C loogikafunktsioon ja $R(C) = \{r\}$ kõigi võimalike rikete hulk selles skeemis. Rikke $r \in R(C)$ olemasolul transformeerub antud skeem C üheks uueks – vigaseks kombinatsioonskeemiks C^r funktsiooniga $y = f^r(X)$. Igat sisendvektorit $t_i = X_i$* võib vaadelda kui testi kombinatsioonskeemile, sisendvektorite jada $T = (t_1, t_2, t_n)$ aga nimetame *testjadaks*.

Ütleme, et testvektor t avastab rikke $r \in R(C)$ antud skeemis siis ja ainult siis kui kehtib $f(X) \neq f^r(X)$.

Riket $r \in R$ nimetatakse *testitavaks* kui eksisteerib vähemalt üks testvektor t, mis avastab rikke r, vastasel juhul on rike r *mittetestitav*.

Kombinatsioonskeemi C nimetatakse *liiaseks*, kui eksisteerib mittetestitav rike $r \in R(C)$. Sellist skeemi on alati võimalik lihtsustada, eemaldades temast vähemalt ühe ventiili või ventiili sisendi. Mittetestitavat riket võib analoogia põhjal samuti nimetada *liiaseks rikkeks*.

Oletame näiteks, et SAF1 rike (konstant 1) JA-ventiili sisendis on mittetestitav. See aga tähendab, et ventiili funktsioon ei muutu antud rikke olemasolul, mistõttu me võime panna sellesse sisendisse permanentselt konstandi 1. Aga n-sisendiga JA-ventiil konstandiga 1 ühes sisendis on ekvivalentne $(n – 1)$-sisendiga JA-ventiiliga, mis on saadud eemaldades

ventiilist konstantse signaaliga sisendi. Analoogiliselt, kui JA-ventiili sisend SAF0 on mittetestitav, siis võib selle JA-ventiili üldse skeemist eemaldada ja asendada konstantse 0-signaaliga.

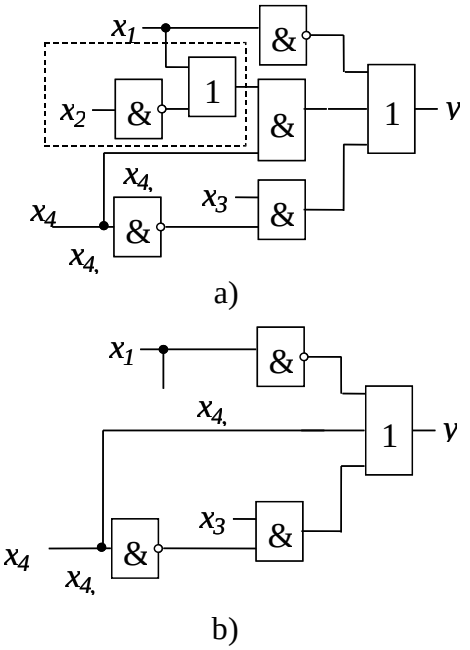
Näide 2-6.

Vaatleme skeemi joonisel 2-13a. Püüdes lahendada *Boole'i differentsiaalvõrrandit*

$$\frac{\partial y}{\partial x_2} = \frac{\partial(\overline{x_1} \vee (x_1 \vee \overline{x_2})x_4 \vee x_3\overline{x_4})}{\partial x_2} = 1,$$

leiame, et

$$\frac{\partial y}{\partial x_2} \equiv 0.$$



Joonis 2-13. *Liiase kombinatsioonskeemi lihtsustamine*

Seega sisend x_2 on skeemis liiane ja ta võib sealt koos invertoriga eemaldada.

Kuna Boole'i tuletis $\partial y/\partial x_2$ on konstantne suvalise x_2 väärtuse korral, siis on mõlemad rikked $x_2 \equiv 0$ ja $x_2 \equiv 1$ selles sisendis mittetestitavad ja ka VÕI elemendi võib skeemist eemaldada, ning selle järel kohe ka järgmise JA-elemendi. Kombinatsioonskeemi funktsioon on nüüd võtnud kuju:

$$y = \overline{x_1} \vee x_{4,1} \vee x_3 \overline{x_{4,2}}$$

ja sellele vastav lihtsustatud kombinatsioonskeem on esitatud joonisel 2-13b. Differentseerides saadud funktsiooni harumuutuja $x_{4,2}$ järgi, saame:

$$\frac{\partial y}{\partial x_{4,2}} = x_1 \overline{x_4} x_3 = 1.$$

Siit näeme, et ehkki rike $x_{4,2} \equiv 1$ on avastatav vektoriga (x_1, x_3, x_4) = 110, ei ole seevastu rike $x_{4,2} \equiv 0$ testitav. Eemaldades nüüd invertori sisendis x_4 ja seejärel ka mittetestitava konstant 1 rikke tõttu invertorile järgneva JA-elemendi sisendi, saame uueks lihtsustatud funktsiooniks:

$$y = \overline{x_1} \vee x_4 \vee x_3 .$$

Liiased mittetestitavad rikked põhjustavad tõelise probleemi testvektorite genereerimisel. Testide genereerimine on protseduur, kus lahendit tuleb otsida kõikvõimalike testvektorite või testjadade hulgast. Mitteliiaste ehk testitavate rikete puhul, me harilikult leiame testi kiiresti, skaneerides enamasti vaid väikest osa kogu otsimisruumist. Liiaste rikete puhul aga tuleb läbida kogu otsinguruum, mis on väga töömahukas, ja kus kõige krooniks selle töömahuka otsingu lõpptulemuseks on teada saamine, et otsitavat testi ei eksisteerigi.

Seetõttu, kui eemaldaksime liiased skeemi osad juba nende projekteerimise käigus, enne testide genereerimist, suudaksime testvektorite sünteesi kiirust märgatavalt tõsta.

2.2.2. Rikete dominantsus- ja ekvivalentsussuhted

Olgu $y = f(X)$ mingi kombinatsioonskeemi C loogikafunktsioon ja $R(C) = \{r\}$ kõigi võimalike rikete hulk selles skeemis.

Rikete ekvivalentsus

Kaks riket $r \in R(C)$ ja $s \in R(C)$ on *funktsionaalselt ekvivalentsed*, siis ja ainult siis kui $f^r(X) \equiv f^s(X)$.

Ütleme, et testvektor *t eristab* kahte riket $r \in R(C)$ ja $s \in R(C)$ siis, kui $f^r(X) \neq f^s(X)$. Selliseid rikkeid nimetatakse *eristatavateks*. Ei ole olemas testi, mis eristaks kahte funktsionaalselt ekvivalentset riket.

Funktsionaalse ekvivalentsuse suhe lahutab kogu rikete hulga $R(C)$ funktsionaalselt *ekvivalentseteks klassideks*. Testide genereerimise aspektist on piisav vaadelda ainult ühte esindajat igast *ekvivalentsusklassist*.

Iga n-sisendilise ventiiliga me võime seostada $2(n + 1)$ üksikut konstantset riket. JA-EI ventiili puhul kõik konstant 0 rikked sisendites ja konstant 1 rike väljundis on funktsionaalselt ekvivalentsed. Kõiki neid rikkeid võib esindada üheainsa rikkega, näiteks väljundi konstant 1 rikkega. Seega iga n-sisendilise ($n>1$) JA-EI ventiili puhul piisab tegeleda testide genereerimise ajal $2(n + 1)$ rikke asemel vaid $n+2$ üksiku konstantse rikkega.

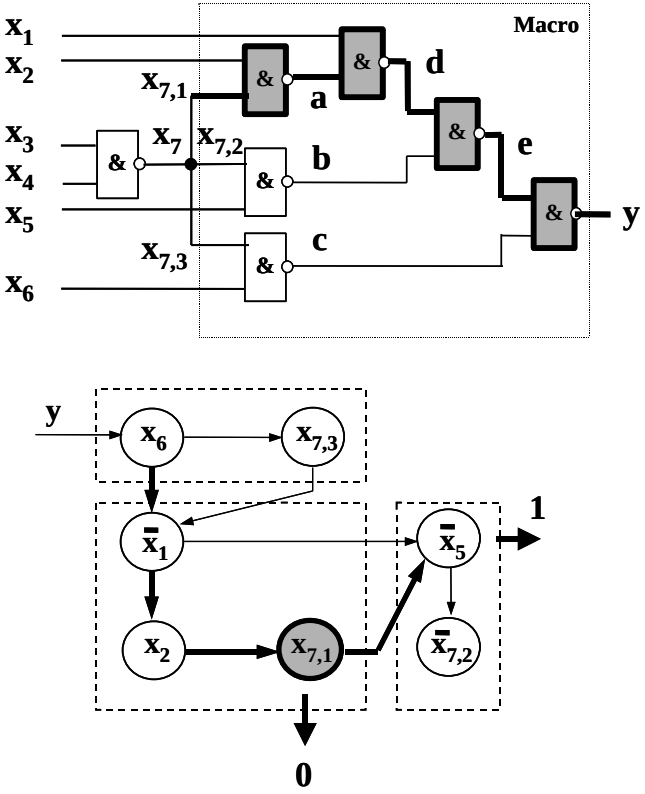
Niisugust oluliste rikete hulga redutseerimise meetodit nimetatakse *ekvivalentsete rikete kollapsiks* (*equivalence fault collapsing*).

Kui lisaks rikete avastamisele on testimise ülesandeks ka rikete lokaliseerimine, tuleb genereerida niisugusi teste, mis mitte üksnes ei avasta rikkeid, vaid suudaksid neid ka eristada. *Täielik rikkeid lokaliseeriv test* suudab eristada igat riket eristatavate rikete hulgast antud skeemis. Täielik rikkeid lokaliseeriv test suudab diagnoosida rikkeid kuni funktsionaalse ekvivalentsuse täpsuseni. Maksimaalse *diagnoosi resolutsiooni* (eristatavuse) määrabki funktsionaalse ekvivalentsuse suhe antud rikete hulgal.

Struktuurselt sünteesitud otsustusdiagrammide konstrueerimisel toimub automaatselt ka ekvivalentsete rikete kollaps: kui iga tipp SSBOD mudelis esindab ühte *signaaliteed* vastavas puukujulises kombinatsioonskeemis, siis igale *tipurikkele* otsustusdiagrammis vastab samuti üks rikete alamhulk ventiilide sisendites tipule vastaval signaaliteel.

Näide 2-7.

Vaatleme kombinatsioonskeemi ja sellele vastavat SSBOD mudelit joonisel 2-14.



Joonis 2-14. *Rikete kollaps SSBOD mudelis*

Joonisel 2-14 on esitatud *puukujuline makro C* kombinatsioonskeemist ja talle vastav SSBOD. Rikete hulk makros on $R(C)$. Igale tipule m tipumuutujaga $x(m)$ selles graafis vastab *signaalitee* $P(m)$ kombinatsioonskeemis, mis algab muutujaga $x(m)$ tähistatud ühendusest ja levib väljundini y. Testida tippu m graafis rikkele $x(m) \equiv e$, $e \in \{0,1\}$ hulgast $R(C)$ tähendab testida rikete alamhulka $F(x(m),e)$ signaaliteel $P(m)$.

Olgu $P(m) = (x(m), x_1, \ldots, x_n)$, kus $x(m)$ on sisendmuutuja signaaliteel $P(m)$ ja $x_1, \ldots, x_n$ on ventiilide väljundmuutujad sellel teel. Siis graafi tipu m rikkega $x(m) \equiv e$ on ekvivalentne järgmine rikete hulk signaaliteel $P(m)$:

$$F(x(m),e) = \{x(m) \equiv e, x_1 \equiv e \oplus INV_1, \ldots, x_n \equiv e \oplus INV_n\}$$

kus $INV_i = 0$ kui invertorite arv sõlmest x_i kuni makro väljundini on paarisarv, ja $INV_i = 1$ vastupidisel juhul.

Näiteks, rike $x_2 \equiv 1$ otsustusdiagrammi tipus x_2 on ekvivalentne järgmise rikete alamhulgaga signaaliteel $P(x_2)$:

$$F(x_2,1) = \{x_2 \equiv 1, a \equiv 0, d \equiv 1, e \equiv 0, y \equiv 1\}.$$

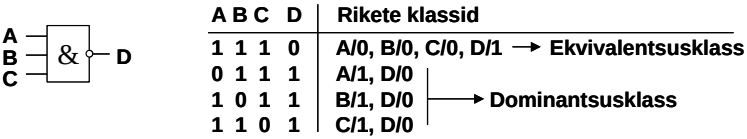
Rikete dominantsus

Kui testimise eesmärgiks on üksnes rikete avastamine, siis lisaks ekvivalentsuse suhtele võib kasutada ka rikete dominantsuse suhet, mis samuti aitab redutseerida käsitlemist vajavate rikete hulka.

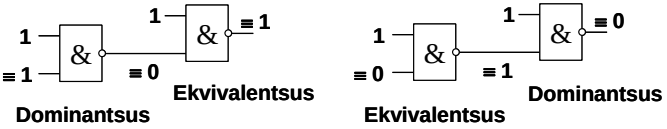
Olgu T_r niisuguste testvektorite hulk, mis avastavad rikke $r \in R(C)$. Rike *r domineerib* rikke $s \in R(C)$ üle siis ja ainult siis kui rikked *r* ja *s* on funktsionaalselt ekvivalentsed testide hulga T_r jaoks.

Kui rike $r \in R(C)$ domineerib rikke $s \in R(C)$ üle, siis iga test *t*, mis avastab rikke *s*, avastab ka rikke *r*. Seetõttu, kui testide genereerimise ülesandeks on ainult *rikete avastamine*, pole oluline käsitleda domineerivaid rikkeid *r*, sest genereerides testvektori rikkele *s*, me automaatselt leiame testi, mis avastab ka domineeriva rikke *r*.

Rikete ekvivalentsus ja dominantsus:



Rikete kollaps:



Joonis 2-15. *Rikete kollaps ekvivalentsus/dominantsuse suhete abil*

Rikete kollaps

Rikete ekvivalentsuse ja dominantsuse suhteid võib kasutada oluliste rikete hulga minimeerimiseks, mida tuleb kasutada testide genereerimisel ja rikete simuleerimisel.

Vaatleme 3 sisendiga JA-EI elementi joonisel 2-15. Tabelis on esitatud ventiili täielik test ja iga testvektori jaoks on määratud ekvivalentsuse- ja dominantsuse klassid. Dominantsusklasside puhul on oluline see, et selles klassis konkreetse testvektori puhul on alati üks rike, mida võib avastada ka mõne teise testvektoriga. Näiteks väljundriket $D \equiv 0$[4] avastab koguni kolm erinevat vektorit. Seepärast just see rike domineeribki temaga seotud olevates dominantsusklassides teiste rikete üle.

Rikete kollapsi reeglid:

1. *Ekvivalentsusreegel*: suvaline rike ekvivalentsusklassis võib saada valitud selle klassi esindajaks, jättes ülejäänud ekvivalentsusklassi rikked vaatluse alt välja.
2. *Dominantsusreegel*: dominantsusklassidest võib üksnes domineerivaid rikkeid vaatlusest välja jätta.

Vaatleme nüüd lihtsat kahe ventiiliga kombinatsioonskeemi joonisel 2-15. Testimisele kuuluvate rikete minimeerimiseks antud skeemis võib nüüd kasutada mõlemat rikete kollapsi reeglit.

Näiteks, testvektori 110 puhul (vasakpoolne skeem) valime ekvivalentsusreegli põhjal väljundventiili sisendrikke $\equiv 0$ klassi esindajaks ja seejärel seades selle rikke dominantsussuhtesse rikkega $\equiv 1$ skeemi sisendis, valime sisendi rikke $\equiv 1$ kõigi rikete esindajaks vaadeldaval signaaliteel.

Testvektori 111 puhul aga (parempoolne skeem) valime dominantsusreegli põhjal väljundventiili sisendrikke $\equiv 1$ klassi esindajaks ja seejärel seades selle rikke ekvivalentsussuhtesse rikkega $\equiv 0$ skeemi sisendis, valime sisendi rikke $\equiv 0$ kõigi rikete esindajaks vaadeldaval signaaliteel.

[4] Rikke $x \equiv e$, $e \in \{0,1\}$, tähistamiseks kasutatakse ka notatsiooni x/e

Vaadeldud näitest tuleneb vahetult üldine rikete kollapsi reegel puukujuliste skeemide jaoks: puukujulistes kombinatsioonskeemides on vaja testide üksnes sisendrikkeid.

Üldistades seda reeglit kombinatsioonskeemide üldjuhtumi jaoks saame järgmise üldreegli.

Rikete kollapsi üldreegel kombinatsioonskeemide jaoks:

Kombinatsioonskeemide testimiseks konstant tüüpi riketele on vajalik ja piisav testida:

- konstantsed rikked skeemi kõigi hargnemispunktide kõikides harudes, ja
- konstantsed rikked skeemi kõigis sisendites.

2.2.3. Rikete maskeerumine

Kordsed rikked

Seni oleme vaadelnud vaid üksikuid rikkeid skeemis. Oleme teadlikult lihtsustanud oma niigi rasket rikete modelleerimisega seotud ülesannet, kuna juba üksikute rikete hulk digitaalsüsteemides võib osutuda käsitlematult suureks, rääkimata siis suvaliste rikete kombinatsioonide arvust.

Olgu skeemis n signaalijuhet, siis üksikute konstantrikete hulk selles skeemis sisaldab $2n$ riket. Kõikvõimalike kordsete rikete hulk sisaldab aga kokku 3^n-1 riket, kus iga juhe võib olla 3-es üksteist välistavas olekus: konstant 0, konstant 1 ja "korras" juhe.

Kui pisut lihtsustada ülesannet ja anda ette nn. "lubatud rikete kordsus" k, s.t. et suurim korraga eksisteerivate rikete arv võib olla k, siis kõigi võimalike rikkesituatsioonide (rikete kombinatsioonide) arv N niisugusel juhul oleks:

$$N = \sum_{j=1}^{k} \{C_j^n\} 2^j ,$$

kus C_j^n on kombinatsioonide arv n-ist j kaupa.

Ülaltoodust näeme, et kordsete rikete arv on väga suur. Seega oleks mõeldamatu nende numereerimine ja käsitlemine vähegi suuremas kombinatsioonskeemis. Küll aga tuleb kordsete rikete võimalust siiski arvestada testide genereerimisel just rikete vastastikkuse maskeerimise ohu vältimiseks.

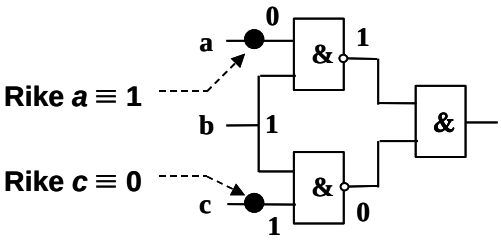
Rikete maskeerumine

Olgu $y = f(X)$ mingi kombinatsioonskeemi C loogikafunktsioon ja $R(C) = \{r\}$ kõigi võimalike rikete hulk selles skeemis. Olgu T_r niisuguste testvektorite hulk, mis avastavad rikke $r \in R(C)$.

Rike $s \in R(C)$ maskeerib riket $r \in R(C)$ testi T_r puhul siis ja ainult siis kui ükski testvektor $t \in T_r$ ei avasta kordset riket $\{r, s\} \subset R(C)$.

Näide 2-8.

Vaatleme skeemi joonisel 2-16, kus on kaks riket $a \equiv 1$ ja $c \equiv 0$, mis maskeerivad teineteist.



Joonis 2-16. *Kahe rikke maskeerumine kombinatsioonskeemis*

Testvektor $t = (a,b,c) = 011$ on ainus võimalik vektor, mis avastaks rikke $c \equiv 0$. Aga see vektor ei avasta riket $c \equiv 0$, kui samaaegselt on skeemis ka teine rike $a \equiv 1$. Seega, võime öelda, et rike $a \equiv 1$ maskeerib testvektori t rakendamisel riket $c \equiv 0$.

Olgu T kõigi võimalike testvektorite hulk ja $T_r \subseteq T$ kõigi nende võimalike testvektorite hulk, mis avastavad rikke $r \in R(C)$.

Me ütleme, et rike $s \in R(C)$ maskeerib riket $r \in R(C)$, siis ja ainult siis kui ükski testvektor $t \in T_r$ ei avasta kordset riket $\{r, s\} \subset R(C)$.

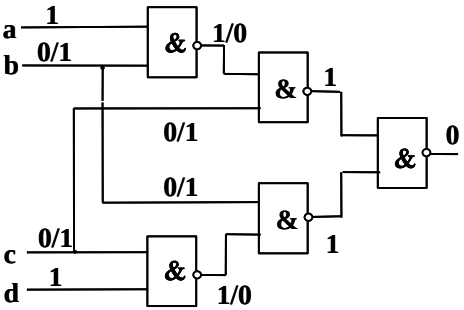
Rikete ringmaskeeruvus

Olgu T *täielik* testvektorite hulk, mis avastavad skeemis C kõik üksikud konstantrikked hulgast $R(C)$. Kordne rike $R' \subset R(C)$ võib aga maskeeruda niisuguse täieliku üksikrikete testi T korral, ehkki kõik rikked $r \in R'$ eraldi oleksid testiga T avastatavad.

Näide 2-9.

Vaatleme skeemi joonisel 2-17. Test T = {1111, 0111, 1110, 1001, 1010, 0101} avastab kõik üksikud konstantrikked selles skeemis. Skeemis on aga kordne rike $R' = \{b \equiv 1, c \equiv 1\}$. Ainus testvektor, mis avastab mõlemat riket üksikult on $1001 \in T$. Mõlema rikke koosesinemisel, aga ei testita enam kumbagi selle vektoriga, sest rike $b \equiv 1$ maskeerib riket $c \equiv 1$ ja rike $c \equiv 1$ maskeerib omakorda riket $b \equiv 1$.

Niisugust maskeerumist nimetatakse *ringmaskeerimiseks* (*circular masking*).



Joonis 2-17. *Rikete ringmaskeerumine kombinatsioonskeemis*

Eelnevatest näidetest on näha, et kordsete rikete olemasolu võib mõjutada testide kvaliteeti. Seetõttu, ehkki me pole suuteliselt käsitlema kõiki võimalikke kordseid rikkeid ja konstrueerima kõigile neile sobivaid testvektoreid, on ometi olemas ka veel teine võimalus – genereerida igale üksikule rikkele testi niiviisi, et see rike saaks avastatud antud testi poolt

suvalise kordse rikete kombinatsiooni korral. Niisugusi meetodeid vaatleme edaspidi.

2.3. Funktsionaalne rikke mudel

Digitaalsüsteemide testide genereerimise efektiivsus on oluliselt sõltuv süsteemi ja selle rikete diagnostilise modelleerimise viisidest. On piisavalt hästi näidatud, et konstantse rikke mudel ei suuda garanteerida piisavat testimise kvaliteeti. Põhjuseks on see, et konstantse rikke mudel ignoreerib tegelikku transitorskeemide käitumist (eriti *CMOS tehnoloogia* korral) ja ei kajasta adekvaatselt reaalseid füüsikalisi defekte transistorskeemides, mis tihtilugu pole esitatavad ei üksikute ega ka kordsete konstantsete riketena loogikatasandil.

Samal ajal füüsikaliste defektide käsitlemiseks rikete simuleerimise tööriistadega, me ometigi vajame loogikatasandi mudeleid, kuna see võimaldab vähendada simuleerimise keerukust (rida füüsikalisi defekte võib modelleerida ühe ja sama loogikatasandi rikkega) ja loogikarikete mudeli kasutamise korral oleme simuleerimisel sõltumatud konkreetsest tehnoloogiast.

Käesolevas peatükis vaatleme meetodit, kuidas modelleerida füüsikalisi defekte üldistatud differentsiaalvõrranditega eesmärgil kujutada neid füüsikaliselt tasandilt loogikatasandile. Vaatleme ja analüüsime mitmeid transitortasandi rikete tüüpe, et veenduda meetodi üldisuses. Defineerime ka uue funktsionaalse rikke mudeli ja näitame, kuidas seda mudelit saab käsitleda kui *universaalset liidest*, mille läbi kujutada rikkeid suvaliselt alumiselt tasandilt kõrgemale. Niisugune rikete kujutamine tasandilt tasandile teeb võimalikuks hierarhilise lähenemisviisi testimisele, mis on ka ainuvõimalik strateegia tänapäevaste üha keerukamaks muutuvate digitaalsüsteemide testimiseks.

2.3.1. Rikete modelleerimine Boole'i differentsiaalvõrranditega

Vaatleme digitaalskeemi ühte skeemisisest komponenti G, mis realiseerib Boole'i funktsiooni $y = f(x_1, x_2, \ldots, x_n)$. Võtame kasutusele täiendava Boole'i muutuja d mingi suvalise füüsikalise defekti tähistamiseks komponendis. Olgu teada ka defektse komponendi (defektiga d) realiseeritav Boole'i funktsioon:

$$y = f^d(x_1, x_2, \ldots, x_n),$$

kus üldjuhul võivad mõned muutujad ka puududa, juhul kui defekt *d* välistab nende mõju komponendi väljundile.

Võtame kasutusele komponendi *G* jaoks *üldistatud parameetrilise funktsiooni* (parameetriks on defekt *d*):

$$y^* = f^*(x_1, x_2, \ldots, x_n d) = \bar{d} f \vee d f^d \qquad (2\text{-}1)$$

mis kirjeldab komponendi käitumist üheaegselt kahel võimalikul viisil – töökorras skeemina ja defektsena (defektiga *d*). Defektse oleku korral on defektimuutuja *d* väärtuseks 1, ja korras skeemi puhul $d = 0$. Teiste sõnadega, $y^* = f^d$ kui $d = 1$, ja $y^* = f$ kui $d = 0$.

Boole'i differentsiaalvõrrandi

$$W^d = \frac{\partial y^*}{\partial d} = 1. \qquad (2\text{-}2)$$

lahend kirjeldab tingimust (kitsendust), mille täitmisel defekti mõju on aktiveertud komponendi väljundisse, s.t. et korras skeem ja defektne skeem annaksid kitsenduse (2-2) täitmisel väljundis erinevad signaalid.

Parameetriline defekti *d* modelleerimine differentsiaalvõrranditega (2-1) ja (2-2) lubaks kasutada kitsendusi $W^d = 1$ nii testide genereerimisel defekti *d* avastamiseks, kui ka rikete simuleerimisel, kontrollimaks kas defekt *d* on antud testvektoriga avastatav. Esimesel juhul tuleks lahendada võrrand (2-2), teisel juhul tuleks arvutada võrrandi (2-2) vasaku poole väärtust.

Et leida tingimus W^d ette antud defekti *d* jaoks, peame kõigepealt leidma defektse komponendi funktsiooni f^d kas loogilise tuletamise teel defektse komponendi jaoks konstrueeritud *ekvivalentskeemist*, või viia läbi defektse komponendi mudeli simuleerimine, või korraldada reaalsed eksperimendid defektse komponendiga, et kindlaks teha defekti mõju komponendi käitumisele.

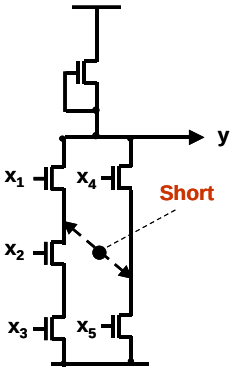
Näide 2-10.

Vaatleme transistorskeemi, mis on esitatud joonisel 2-18 ja mis realiseerib funktsiooni

$$y = x_1 x_2 x_3 \vee x_4 x_5$$

Lühis skeemis, nii nagu näidatud joonisel, muudab skeemi funktsiooni järgmiselt:

$$y^d = (x_1 \vee x_4)(x_2 x_3 \vee x_5)$$



Joonis 2-18. *Lühisega transistorskeem*

Võttes lühise tähistamiseks kasutusele muutuja d, konstrueerime antud defekti analüüsimiseks komponendi üldistatud funktsiooni ja differentseerime seda muutuja d suhtes. Viies läbi vajalikud lihtsustused, leiame differentsiaalvõrrandi lahendid, mis oleksidki defekti avastamise tingimusteks transistorskeemi väljundis:

$$\frac{\partial y^*}{\partial d} = \frac{\partial((x_1x_2x_3 \vee x_4x_5)\overline{d} \vee (x_1 \vee x_4)(x_2x_3 \vee x_5)d)}{\partial d} =$$

$$= \frac{\partial(x_1x_2x_3(\overline{d} \vee d) \vee x_4x_5(\overline{d} \vee d) \vee (x_1x_5 \vee x_2x_3x_4)d)}{\partial d} =$$

$$= \frac{\partial(x_1x_2x_3 \vee x_4x_5 \vee (x_1x_5 \vee x_2x_3x_4)d)}{\partial d} =$$

$$= \overline{x_1x_2x_3}\,\overline{x_4x_5}(x_1x_5 \vee x_2x_3x_4)$$

$$= x_1\overline{x_2}\,\overline{x_4}x_5 \vee x_1\overline{x_3}\,\overline{x_4}x_5 \vee \overline{x_1}x_2x_3x_4\overline{x_5} = 1$$

Lahenditeks on järgmine tingimuste hulk:

$$T = \{10x01, 1x001, 01110\}.$$

Igatüht nendest tingimustest võiks kasutada kui testvektorit komponendi sisendis antud defekti avastamiseks.

Toodud näidet võiks veel vaadelda kui kontranäidet, tõestamaks konstantsete rikete mudeli ebaadekvaatsust. Vaatleme järgnevat tabelit.

Tabel 2-1. *100%-lise KR testi ja defekti testi võrdlus*

No	100% KR test					Defekti test				
	x_1	x_2	x_3	x_4	x_5	x_1	x_2	X_3	x_4	x_5
1	1	1	1	0	-	1	0	-	0	1
2	0	-	-	1	1	1	-	0	0	1
3	0	1	1	0	1	0	1	1	1	0
4	1	0	1	1	0					
5	1	1	0	-	0					

Tabelis 2-1 on toodud 5 testvektorit, mis avastavad 100%-liselt kõik konstantrikked (KR) antud skeemis (vasakul) ja testvektorid, mis on genereeritud spetsiaalselt antud defekti jaoks (paremal). KR testvektorite jaoks on värvitud ruutudega näidatud, millised rikked konkreetne testvektor avastab. 0(1) värvitud ruudus näitab, et reale vastav testvektor avastab veerule vastavas sisendis konstantrikke 1(0).

Tabelist näeme, et ükski kolmest testvektorist, mis avastavad antud lühise, ei tulene KR testi vektoritest (ühtegi neist ei saaks tuletada, valides sobivalt muutujate vabu ehk määramata väärtusi).

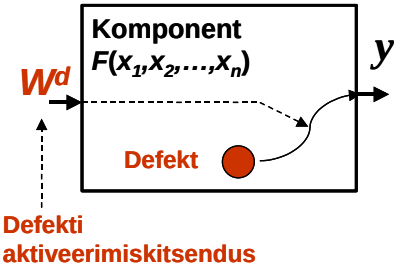
Näitest järeldub seega, et väga hea konstantsete rikete avastamiseks mõeldud test ei pruugi sugugi olla enam piisavalt hea, kui "lubatud" rikete klassi laiendada suvalistele füüsikalistele defektidele. Siit aga tuleneb vajadus üldisema rikete modelleerimise meetodi järele, võrreldes sellega, mida võimaldab konstantrikete mudel.

2.3.2. Füüsikaliste defektide kujutamine loogikatasandile

Eelpool kirjeldatud Boole'i differentsiaalvõrrandile toetuv rikete modelleerimise meetod kujutab endast üldist lähenemisviisi suvaliste defektide kujutamiseks füüsikaliselt tasandilt loogikatasndile. Meetodi abil võib suvalist füüsikalist defekti kirjeldada kitsendusena $W^d = 1$, mis tuleb täita defekti aktiveerimiseks komponendi väljundisse (joonis 2-19).

Väära signaali ilmnemist komponendi väljundis võime Boole'i differentsiaali põhimõttel kirjeldada tingimusena $dy = 1$.

Võttes nüüd kasutusele *funktsionaalse rikke mudeli* mõiste, võime kirjeldada defekti d paariga (dy, W^d). Rike avaldub nüüd puhtalt komponendi funktsiooni kaudu: rakendades komponendi sisenditele tingimusega $W^d = 1$ määratud vektorit. Riket kas avastatakse, kui $dy = 1$, või ei avastata, kui $dy = 0$. Komponendi madala taseme *transistorskeemi* struktuur pole enam oluline, komponenti ennast võib vaadelda musta kastina. Põhimõtteliselt pole ka tingimuste W^d leidmisel vaja teada komponendi struktuuri, kui on teada tema defektne funktsioon f^d.



Joonis 2-19. *Defekti aktiveerimine komponendis*

Teisest küljest, Boole'i differentsiaalvõrrandit (2-2) võib interpreteerida ka kui füüsikalise *defekti kujutamist* füüsikaliselt tasandilt loogikatasandile: füüsikalise defekti asemel võib loogikatasandil käsitleda viga $dy = 1$ (ehk konstantriket $y \equiv 1$), juhul kui kitsendus $W^d = 1$ on täidetud.

Järgmised näited demonstreerivad meetodi kasutatavust mitmete transistorskeemi defektide kujutamisel loogikatasandile.

Lühised (stuck-on) transistorskeemis.

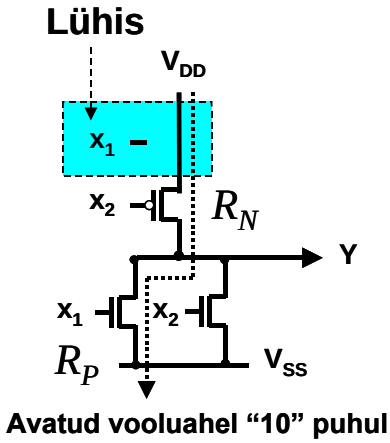
Näide 2-11.

Transistortasemel esitatud VÕI-EI elemendi (joonis 2-20) käitumist vaadeldava lühisrikke korral ei saa kirjeldada rangelt loogiliselt. Sisendvektor "10" produtseerib voolu juhtiva ahela sõlmest V_{DD} sõlmeni V_{SS}, ja vastav pingenivoo väljundsõlmes *Y* ei ole võrdne ei V_{DD} ega V_{SS} – ga, vaid selle asemel kehtib funktsioon *pinge jagunemisest* avatud transistoride *sisetakistustel* voolu juhtivas ahelas.

$$V_Y = \frac{V_{DD} R_P}{(R_P + R_N)}.$$

Sõltuvalt sisetakistuste suhtest ja vastavast väljundisse *y* ühendatud järgnevate komponentide ümberlülitumise *lävipingetest*, defekt võib avalduda veana väljundis *d*, aga võib ka mitte avalduda. Tähistame sümboliga *Z määramatu signaali* väljundis. Siis võib defektse komponendi funktsiooni kirjeldada järgmiselt:

$$y^d = \overline{x_1}\,\overline{x_2} \vee x_1 \overline{x_2} Z.$$



x_1	x_2	y	y^d
0	0	1	1
0	1	0	0
1	0	0	Z: V_Y/I_{DDQ}
1	1	0	0

Joonis 2-20. *Lühisrike (stuck-on) transistorskeemis*

Kui nüüd $x_1\overline{x_2}=1$ siis $y^d = Z$. Kasutades nüüd valemeid (2-1) ja (2-2) saame:

$$y^* = \overline{d}(\overline{x_1 \vee x_2}) \vee d(\overline{x_1}\,\overline{x_2} \vee x_1\overline{x_2}Z)$$

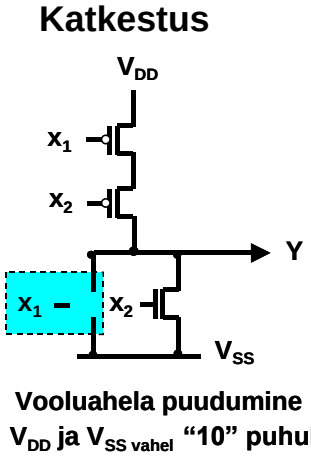
$$W^d = \partial y^* / \partial d = x_1\overline{x_2}Z = 1.$$

Võrrandist lähtub, et defekti aktiveerimiseks on vajalikud tingimused $x_1 = 1, x_2 = 0$, aga defekti avastamiseks on tarvilik ka Z = 1. Viimase tingimuse täitmine sõltub küll juba konkreetsest elektrilisest skeemist.

Katkestused (stuck-open) transistorskeemis

Näide 2-12.

Vaatleme katkestusriket VÕI-EI elemendis joonisel 2-21. Mõningate testvektorite puhul ilmneb, et vooluahelat V_{DD} ja V_{SS} vahel ei tekigi. Selle tulemusena väljundsõlm säilitab oma eelmise loogilise väärtuse. Kombinatoorne element on hakanud käituma kui *dünaamiline mäluelement*.



x_1	x_2	y	y^d
0	0	1	1
0	1	0	0
1	0	0	Y'
1	1	0	0

Joonis 2-21. *Katkestusrike (stuck-open, stuck-off) transistorskeemis*

Defektse ventiili funktsioon on:

$$y^d = \overline{x_1}\,\overline{x_2} \vee x_1\overline{x_2}y'$$

kus *y'* vastab väljund signaali väärtusele, mis oli salvestatud defektse ventiili väljundis eelmisel ajahetkel. Kasutades nüüd avaldisi (2-1) and (2-2) saame:

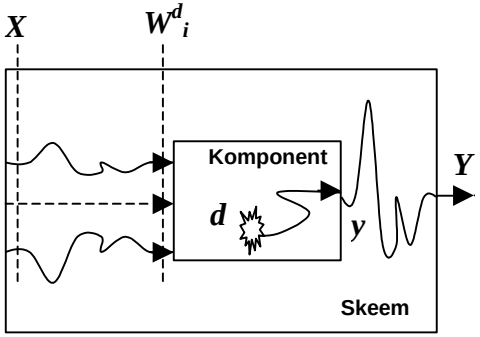
$$y^* = \overline{d}(\overline{x_1 \vee x_2}) \vee d(\overline{x_1}\,\overline{x_2} \vee x_1\overline{x_2}y') =$$
$$= \overline{x_2}(\overline{x_1} \vee dy')$$

$$W^d = \partial y^* / \partial d = x_1\overline{x_2}y' = 1.$$

Siit aga järgneb, et defekti aktiveerimiseks vajalikud tingimused on: $x_1 = 1, x_2 = 0, y' = 1.$ Teiste sõnadega, et testida antud katkestusriket, oleks vaja rakendada 2-vektorilist testjada: kõige pealt vektorit "00", et viia väljund olekusse *y* = 1, ja seejärel vektorit "10", et avastada defekt.

Defektide esitamine defektide tabelina

Üldjuhul võib ühel ja samal defektil *d* olla terve hulk erinevaid *aktiveerimistingimusi* W^d_i, mis leitakse differentsiaalvõrrandi kõigi lahendite hulgana. Olgu antud component C_y, mille väljundiks on *y*, koos kõigi võimalike defektide hulgaga *D*.



***Joonis 2-22**. Defekti aktiveerimine kogu skeemi tasandil*

Olgu $W^{F,d}(y) = \{W^d_i\}$ kõigi loogikatingimuste hulk, mis aktiveeriksid defekti $d \in D$ väljundisse y, ning $W^F(y) = \{W^{F,d}(y)\} \subseteq W(y)$ loogikatingimuste hulk, mis kokku aktiveeriksid kõik defektid hulgast D väljundisse y. Siin $W(y)$ on kõigi võimalike komponendi C_y sisendvektorite hulk.

Näiteks joonisel 2-18 toodud defekti d avastamiseks väljundis y võiks kasutada kolme erinevat tingimust (vt. Tabel 2-1)

$$W^{F,d}(y) = \{10x01, 1x001, 01110\}.$$

Arvestades seda, et mõned sisendmuutujad on siin määramata, on kõigi võimalike lahendite ehk aktiveerimistingimuste arv antud juhul koguni 5. Oluline oleks kõik need võimalused testide generaatori jaoks ette anda, sest mitte igaüks neist ei pruugi olla lahenduv terve skeemi tasandil, s.t. mitte igaühe jaoks nendest ei pruugi eksisteerida skeemi sisendvektor.

Joonisel 2-22 on illustreeritud testi genereerimise protsess defekti d jaoks kogu skeemi ulatuses. Kõigepealt valitakse üks tingimus $W^d_i \in W^{F,d}(y)$ defekti aktiveerimiseks komponendi väljundisse y. Seejärel lahendatakse vastavad Boole'i võrrandid tingimuse $W^d_i = 1$ rahuldamiseks skeemi sisenditel ning aktiveeritakse *veasignaal* komponendi väljundist y skeemi väljundisse Y. Nende protseduuride tulemusena leitakse defekti d avastav testvektor X skeemi sisendis. Võib juhtuda, et sellist vektorit ei eksisteerigi üldse. Siis tuleks valida mingi teine tingimus $W^d_i \in W^{F,d}(y)$ ja korrata sama protseduuri. Kui testvektorit ei leita mitte ühegi tingimuse jaoks hulgast $W^{F,d}(y)$, tuleb defekti d lugeda *funktsionaalselt liiaseks* ning tema testimine polegi oluline antud skeemis.

Loogikatingimuste hulka

$$W^F(y) = \{W^{F,d}(y)\} \subseteq W(y)$$

võib esitada *defektide tabelina* ehk maatriksina $DT = \|w_{i,j}\|$, kus $w_{i,j} = 1$, kui defect i on avastatav komponendi lokaalse sisendvektoriga j, ning $w_{i,j} = 0$, kui defect i ei ole avastatav vektoriga j. Niisugune defektide esitus loogikaelemendile AND2,2/NOR2 funktsiooniga y = $\overline{AB \vee CD}$ on esitatud Tabelis 2-2.

Tabel 2-2. *Defektide tabel loogikaelemendile AND2,2/NOR2*

i	Fault d_i	Erroneous function f^{di}	Input patterns t_j															
			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	B/C	not((B*C)*(A+D))				1								1	1	1		
2	B/D	not((B*D)*(A+C))				1								1	1		1	
3	B/N9	B*(not(A))	1	1	1					1	1	1	1					
4	B/Q	B*(not(C*D))	1	1	1						1	1	1		1	1	1	
5	B/VDD	not(A+(C*D))									1	1	1					
6	B/VSS	not(C*D)													1	1	1	
7	A/C	not((A*C)*(B+D))				1				1					1	1		
8	A/D	not((A*D)*(B+C))				1				1					1		1	
9	A/N9	A*(not(B))	1	1	1		1	1	1					1				
10	A/Q	A*(not(C*D))	1	1	1		1	1	1						1	1	1	
11	A/VDD	not(B+(C*D))					1	1	1									
12	C/N9	not(A+B+D)+(C*(not((A*B)+D)))		1			1	1			1	1						
13	C/Q	C*(not(A*B))	1	1		1	1	1		1	1	1		1				
14	C/VSS	not(A*B)				1				1				1				
15	D/N9	not(A+B+C)+(D*(not((A*B)+C)))			1		1		1		1		1					
16	D/Q	D*(not(A*B))	1		1	1	1		1	1	1		1	1				
17	N9/Q	not((A*B)+(B*C*D)+(A*C*D))				1												
18	N9/VDD	not((C*D)+(A*B*D)+(A*B*C))													1			
19	Q/VDD	SA1 at Q				1				1				1	1	1	1	1
20	Q/VSS	SA0 at Q	1	1	1		1	1	1		1	1	1					

Igale skeemikomponendile *komponentide teegis* lisatakse täiendav *diagnostiline informatsioon* niisuguse defektide tabeli näol. Neid tabeleid komponentide teegis saab kasutada *defektipõhisel testide genereerimisel* ja ka testide *diagnostilise analüüsi* ehk defektide avastamise protsendi arvutamiseks.

Tabelit 2-2 analüüsides on kerge märgata, et kõikide defektide avastamiseks ei olegi vaja kõiki sisendvektoreid kasutada. Näiteks, kõigi 20 defekti avastamiseks piisab vaid 4-st vektorist: 3, 6, 8 ja 12. Seega elemendi defekte võiks kirjeldada kompaktselt ka niisuguse *funktsionaalse rikke mudelina*:

$$W^F(y) = \{W^{F,d}(y)\} = \{0011, 0110, 1000, 1100\}.$$

Nagu juba eelpool mainitud, skeemi tasandil teste genereerides võib osutuda, et mõni nendest vektoritest pole realiseeritav, mis juhul tuleks kõigi defektide katmiseks Tabelis 2.2 hoopis teisi selles mudelis mitte sisalduvaid sisendvektoreid kasutada.

2.3.3. Ühendusahelate defektide kujutamine loogikatasandile

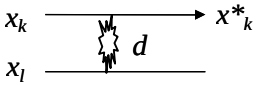
Vaatleme digitaalskeemis ühte komponenti *C*, mida kirjeldab Boole'i funktsioon $y = f(x_1, x_2, \ldots, x_n)$. Kirjeldame *komponendi lähiümbrust*, kus on võimalik lühiste tekkimine komponendi ja ümbruse vahel, signaalijuhtmete hulgana $E_c = \{x_{n+1}, \ldots, x_p\}$. Võtame kasutusele sama täiendava Boole'i muutuja *d* füüsikaliste defektide modelleerimiseks alamskeemis (C,E_c) komponendi *C* ja tema lähiümbruse E_c vahel, mis võiksid mõjutada signaali väärtust komponendi väljundis *y*. Olgu teada defektse alamskeemi (C,E_c) funktsioon:

$$y = f^d(x_1, x_2, \ldots, x_n, x_{n+1}, \ldots x_p). \qquad (2\text{-}3)$$

Võtame kasutusele komponendi *C* ja lähiümbrusega E_c määratud alamskeemi (C,E_c) kirjeldamiseks defekti d võimaliku olemasolu korral *üldistatud parameetrilise funktsiooni* (parameetriks on defekt *d*):

$$y^* = f^*(x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_p, d) = (\overline{d} \wedge f) \vee (d \wedge f^d)$$

mis kirjeldab alamskeemi käitumist üheaegselt kahel võimalikul viisil – töökorras skeemina ja defektsena (defektiga *d*). Defektse oleku korral on defektimuutuja *d* väärtuseks 1, ja korras skeemi puhul $d = 0$. Teiste sõnadega, $y^* = f^d$ kui $d = 1$, ja $y^* = f$ kui $d = 0$. Differentsiaalvõrrandi (2-2) lahendid kirjeldavad tingimusi (kitsendusi), mille korral defekti *d* mõju on aktiveeritud väljundisse *y*.



Joonis 2-23. *Lühis kahe juhtme vahel*

Näide 2-13.

Vaatleme lühist kahe juhtme x_k ja x_l vahel joonisel 2-23. Defektne funktsioon juhtme x_k *lühisdefekti d* vastavalt JA-tüüpi lühismudelile oleks $x^d{}_k = x_k x_l$. Üldistatud funktsiooni saame nüüd kujul

$$x_k^* = \overline{d}x_k \vee dx_k^{\,d} = \overline{d}x_k \vee d(x_k x_l)$$

funktsioonina

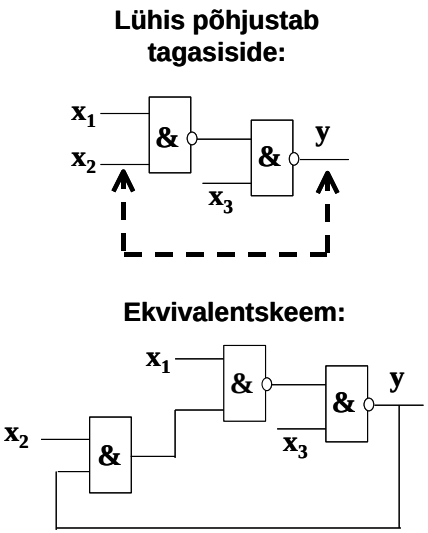
$$x_k^* = f(x_k, x_l, d)$$

nii *defektimuutujast d*, kui ka lähiümbruse muutujast x_l, mis kirjeldab kahe juhtmepaari käitumist nii rikke korral (d = 1) kui ka rikke puudumisel (d = 0).

Boole'i differentsiaalvõrrandi

$$W^d = \partial x_k^* / \partial d = x_k \overline{x_l}$$

lahend kirjeldab tingimust, mille täitmise korral defekt *d* on aktiveeritud juhtmele (on jälgitav juhtmel) x_k (joonis 2-23). Tingimus $W^d = x_k \overline{x_l} = 1$ tähendab: selleks et testida lühist juhtmete x_k ja x_l vahel jälgides signaale juhtmel x_k meil tuleb saata juhtmele x_k signaal 1 ja juhtmele x_l signaal 0.



Joonis 2-24. *Lühis, mis põhjustab tagasiside*

Näide 2-14.

Lühis kahe juhtme x_k ja x_l vahel joonisel 2-24 põhjustab tagasiside. JA-tüüpi lühisriket modelleeriv *ekvivalentskeem* on esitatud samuti joonisel 2-24. *Üldistatud parameetriline funktsioon*, mis kirjeldab skeemi käitumist nii defekti puudumisel kui ka selle olemasolul, võtab järgmise kuju:

$$y^* = \overline{d}(x_1 x_2 \vee \overline{x_3}) \vee d(x_1 x_2 y \vee \overline{x_3}) = x_1 x_2 (\overline{d} \vee y')\overline{x_3}$$

Boole'i differentsiaalvõrrandi

$$W^d = \partial y^* / \partial d = x_1 x_2 x_3 \overline{y'} = 1$$

lahend kirjeldab tingimust (kitsendust), mille puhul defekt *d* on aktiveeritud väljundisse *y* (joonis 2-24). Apostroof *y* juures tähendab, et *y* väärtus kuulub eelmise ajahetke (takti) juurde.

Tingimus

$$W^d = x_1 x_2 x_3 \overline{y'} = 1$$

tähendab, et lühise d testimiseks on vaja 2-st vektorist koosnevat *testjada*: esiteks, vajame vektorit, mis produtseerib skeemi väljundis väärtuse $y = 0$ (näiteks, omistades $x_3 = 0$), seejärel tuleks defekti avastamiseks rakendada vektorit $x_1 = 1$, $x_2 = 1$, $x_3 = 1$.

Käesolev näide demonstreerib seda, et defektide testimiseks vajalikud tingimused võivad üldjuhul puudutada mitut ajahetke, mille tulemusel tuleb genereerida mitte üks vektor, vaid mitmest testvektorist koosnev testjada. Samuti näeme siit, et *Boole'i differentsiaalvõrrandi* lahendamisel põhinev meetod võimaldab universaalselt käsitleda ühtselt nii kombinatoorseid lühiseid kui ka võrdlemisi keerukat sekventsiaalsete lühisdefektide juhtu, kus lühisest tingituna kombinatsioonskeem muutub järjestikskeemiks (või antud järjestikskeemi olekute arv kasvab).

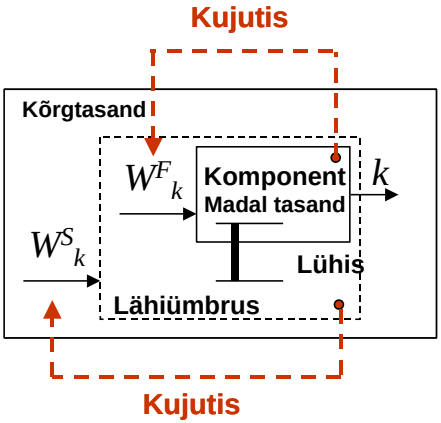
Funktsionaalse rikke mudelit, mis on esitatav paarina (dy, W^d), võib vaadelda, esiteks, meetodina suvaliste füüsikaliste defektide kujutamiseks füüsikaliselt tasandilt loogikatasandile ja teiseks, universaalse meetodina rikete modelleerimiseks testide genereerimise ja rikete simuleerimise hierarhilises käsitluses. Tingimusi W^d defektide *d* aktiveerimiseks võib seejuures kasutada kitsendustena kõrgematel (loogika või registersiirete)

tasanditel kas testide genereerimiseks või rikete simuleerimiseks osutamata tähelepanu madalama taseme rikete (nt. defektide) füüsikalisele olemusele.

2.3.4. Rikete hierarhiline esitus

Rikete defineerimise meetod mudeli (dy,W^d) kaudu võimaldab meil unifitseerida skeemi (või süsteemi) komponentide ja selle lähiümbruse *diagnostilist modelleerimist*, pööramata tähelepanu ei komponendi ega ka selle lähiümbruse struktuursetele omadustele. Mõlemal juhul mudel (dy,W^d) kirjeldab, kuidas madalama tasandi defekti d (kas defekti komponendis või selle lähiümbruses) tuleks aktiveerida kõrgemal tasandil sõlmeni y.

Kitsendusi W^d võib kasutada nii hierarhilisel testide genereerimisel kui ka rikete simuleerimisel. Vaatleme sõlme k joonisel 2-25 kui komponendi M_k väljundit, millele vastab muutuja y_k. Seame sõlmega k vastavusse rikete hulga $R_k = R^F_k \cup R^S_k$, kus R^F_k on rikete alamhulk komponendis M_k väljundiga y_k ja R^S_k on komponendiga M_k seotud struktuursete rikete (defektide) alamhulk komponendi M_k lähiümbruses.

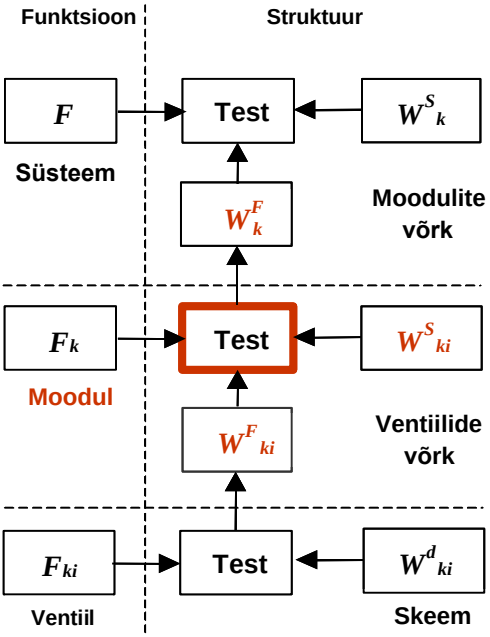


Joonis 2-25. *Lühis, mis põhjustab tagasiside*

Olgu $W^d \in W(y_k)$ tingimus, mille täitmise korral rike $d \in R_k$ muudab muutuja y_k väärtust $(dy_k = 1)$. Olgu $W^F_k \subset W(y_k)$ tingimuste hulk, mis

aktiveerivad defekte $d \in R^F_k$ ja $W^S_k \subset W(y_k)$ nende tingimuste hulk, mis aktiveerivad defekte $d \in R^S_k$.

Kasutades tingimusi W^F_k and W^S_k seame üles rikete kujutised madalamalt tasandilt kõrgemale tasandile hierarhilise testide genereerimise eesmärgil, aga samuti ka vastupidi, kõrgemalt tasandilt madalamale tasandile hierarhilise rikete simuleerimise või diagnoosi eesmärgil.



Joonis 2-26. *Hierarhiline testimise kontseptsioon*

Et kujutada rikkeid $d \in R_k$ testide genereerimise eesmärgil madalamalt tasandilt kõrgemale muutuja y_k differentsiaaliks dy_k on vaja leida lahend võrrandile $W^d = 1$. Teiste sõnadega, kui kitsendus $W^d = 1$ on täidetud, siis defekti $d \in R_k$ olemasolu korral avaldub see signaali y_k muutusena $dy_k = 1$.

Rikete simuleerimisel (või rikete diagnoosil) muutuja y_k väärat väärtust ($dy_k = 1$) võib formaalselt interpreteerida implikatsioonina

$$dy_k \rightarrow d_1 W^{d1} \vee d_2 W^{d2} \vee \ldots \vee d_n W^{dn} \qquad (2\text{-}4)$$

kus $\forall j = 1,2,\ldots n$: $d_j \in R_k$. Teiste sõnadega, kõrgtasandi sündmusele $dy_k = 1$, seatakse vastavusse madalama tasandi sündmus d_j siis, kui tingimus $W^{dj} = 1$ on täidetud.

Hierarhilise testimise käsitluse eesmärgil peaksime konstrueerima iga komponendi M_k jaoks skeemis *rikete nimistu* R_k koos vastavate kitsendustega W^d iga rikke jaoks $d \in R_k$. Kitsenduste hulga W^{F_k} võib genereerida madalamal tasandil kõigile komponendi riketele $d \in R^{F_k}$. Kitsenduste hulga W^{S_k} struktuursete rikete jaoks $d \in R^{S_k}$ komponendi lähiümbruses võib genereerida Boole'i differentsiaalanalüüsi abil eelmistes punktides kirjeldatud meetodil.

Joonisel 2-26 on illustreeritud hierarhilise testimise kontseptsiooni kolme tasandi "süsteem – moodul – ventiil" vahel, nagu eelnevalt kirjeldatud. Funktsionaalse lähenemise korral kasutatakse informatsiooni ainult vaadeldava tasandi objekti funktsionaalse käitumise kohta. Struktuurse lähenemisviisi puhul on testimise objektideks rikked struktuurse võrgu komponentides ja nende vahelistes ühendustes.

2.4. Kõrgtasandi rikete mudelid

Et kiirendada testide simuleerimist ja rikete katte hindamist on välja töötatud kõrgtasandi (funktsionaalse või käitumusliku tasandi) rikkemudeleid.

Kõrgtasandi rikkemudelid väljendavad madala tasandi füüsikaliste defektide mõju süsteemi käitumisele kõrgematel funktsionaalsetel tasanditel – mikrotehete või operatsioonide täitmisel registersiirete tasandil või arvuti käskude täitmisel käsusüsteemide arhitektuursel tasandil. Kõrgtasandi rikkemudelit loetakse heaks, kui tema abil genereeritud testidel on hea konstantsete rikete või füüsikaliste defektide kate (avastamise võime).

Rikete kõrgtasandil modelleerimise põhiidee seisneb süsteemi kõrgtasandi kirjelduse modifitseerimises rikete sisseviimisega nii, et süsteemi esialgsest kirjeldusest saadaks vigaseid versioone. Niisugust lähenemisviisi nimetatakse *mudeli "häirimiseks"* (*model perturbation*) [Gup 85]. Mudelit saab "häirida" mitmel viisil, näiteks, tõeväärtustabeli modifitseerimisega, mikrotehete modifitseerimisega jne. Nimetatud idee on

realiseeritud ühel või teisel viisil erinevates kõrgtasandi rikete mudelites, mis on väljatöötatud spetsiaalselt näiteks mikroprotsessoritele [Tha 80], pisut üldisemal moel regitersiirete tasandil esitatud süsteemidele [She 88] või VHDL keeles kirjeldatud digitaalsüsteemidele [Gho 91] jne. On ka tehtud katseid töötada välja eri kõrgtasandi rikete mudeleid spetsiaalsetele süsteemikomponentidele nagu dekooderid, multiplekserid, mälud, programmeeritavad maatriksid jne.

Kõrgtasandi rikete mudelid võivad olla otsesed ja kaudsed [Abr 95]. *Otsene rikke mudel* identifitseerib iga rikke eraldi ja iga rike selles mudelis on üheks konkreetseks testide genereerimise eesmärgiks. *Kaudne rikke mudel* identifitseerib rikete klasse, kus rikked omavad mingeid sarnaseid omadusi, nii et kõiki ühe klassi rikkeid saaks käsitleda sarnaste protseduuride abil. Kaudse mudeli eeliseks on see, et ta ei nõua rikete otsest numereerimist klassi sees.

Enamus kõrgtasandi rikkeid, mida me selles peatükis käsitleme võib samastada nn. *adresseerimise rikete* (*adressing faults*) [Abr 95] tüübiga. Iseloomulikumateks näideteks on: sõna adresseerimine mälus, registri valik vastavalt konkreetse välja sisule protsessori *käsuvormingus*, tehtekoodi dekodeerimine, et kindlaks teha, millist käsku on vaja täita arvutis jne. Üldiseks põhimõtteks selles mudelis on kasutada *n*-bitist aadressi, et valida ühte 2^n võimalikust objektist. Millal iganes üks objekt *i* on valitud, adresseerimise rike võib tähendada ühte kolmest võimalusest:

- ei valita ühtegi objekti;
- valitakse objekt *j* objekti *i* asemel;
- valitakse objekt *j* lisaks objektile *i*;

Veelgi üldisemalt: võidakse valida üks hulk objekte $\{j_1, j_2, \ldots, j_k\}$ objekti *i* asemel või lisaks objektile *i*.

Oluliseks iseloomujooneks adresseerimisrikete mudeli puhul on kontrollida, et adresseeritud funktsioon täidetakse ja et selle kõrval ei toimuks ka muid tegevusi, mis pole ettenähtud. Seda fundamentaalset aspekti ignoreerivad tihti mitmesugused teised heuristilised rikete mudelid.

2.4.1. Mikroprotsessori funktsionaalne rikete mudel

Väga levinud on lähenemisviis, kus mikroprotsssori erinevate alamsüsteemide nagu andmetöötluse alamsüsteemi (operatsioonautomaati) ja juhtseadme (juhtautomaadi) jaoks kasutatakse erinevaid rikete mudeleid

[Tha 80, Abr 95]. Kuna kõrgtaseme operatsioonid toimuvad kõrgtasandil, mis on hästi jälgitav inimese poolt, siis anomaaliate puhul süsteemi käitumises tuleks rääkida pigem *vigadest* kui riketest, mis vastab ka paremini nende sõnade igapäevasele tähendusele eesti keeles. Säilitamaks aga eespool defineeritud suhet sõnade vahel - defekt, rike, viga - kus rike tähendab defekti mudelit, viiakse protsessori funktsioonide "materialiseerimise" eesmärgil tinglikult sisse nn. *mehhanismi* mõiste. Mehhanismiks loetakse mingit tinglikku osa aparatuurist, mis realiseerib või on vastutav teatavate funktsioonide täitmise eest.

Kõrgtasandi rikete mudeli konstrueerimise seisukohalt võib mikroprotsessorit jagada järgmisteks mehhanismideks: registrite dekodeerimise, käskude dekodeerimise, andmete säilitamise, andmete edastamise ja andmete töötlemise mehhanismid.

Niisiis, kõrgtasandi rikked, mis mõjutavad mikroprotsessori käitumist võib jagada järgmisteks konkreetsete mehhanismidega seotud rikete klassideks:

- *adresseerimise rikked*, mis mõjutavad registrite dekodeerimise mehhanismi,
- adresseerimise rikked, mis mõjutavad käskude dekodeerimise ja mikrotehete järjestamise mehhanisme,
- rikked andmete säilitamise mehhanismis,
- rikked andmete edastamise mehhanismis,
- rikked andmete töötlemismehhanismis.

Rikked registrite dekodeerimise mehhanismis

Registrite dekodeerimise vigade põhjusteks võivad olla defektsed multiplekserid või defektsed demultiplekserid.

Kui on antud mingi *andmeallika* aadress, siis multiplekseris on võimalikud järgmised anomaaliad, mis moodustavad registrite dekodeerimise mehhanismi rikete mudeli:

F1: ei valitagi allikat,

F2: valitakse vale allikas,

F3: allikaid valitakse rohkem kui üks, kusjuures multiplekseri väljund määratakse JA või VÕI tehtega (sõltuvalt *tehnoloogiast*) üle kõigi andmesõnade, mis on valitud.

Demultiplekserite puhul on aga järgmised *käitumisvead* võimalikud, mis lisanduvad rikete mudelisse:

F4: ei valitagi sihtkohta;

F5: valitud sihtkoha asemel või sellele lisaks valitakse üks teine või veelgi rohkem sihtkohti.

Rikked käskude dekodeerimise mehhanismis

Mikroprotsessori käsku *I* võib vaadelda kui ühte *mikrokäskude* jada, kus iga mikrokäsk väljastab juhtsignaale, mille toimel realiseeruvad paralleelselt mikrotehted. *Mikroteheteks* on elementaarsed andmesiirded või tehted andmetega.

Adresseerimise vead, mis mõjutavad käskude täitmist või mikrotehete järjestust, võivad kutsuda esile järgmisi effekte (käskude dekodeerimise mehhanismi rikete mudel):

F6: üks või enam mikrotehet jäävad aktiveerimata käsu *I* poolt;

F7: mikrotehteid aktiveeritakse ekslikult;

F8: õigete mikrokäskude asemel või nendele lisaks aktiveeritakse komplekt mikrokäske, mida ei peaks aktiveerima.

Kirjeldatud rikete mudel on üldine, kuna ta võimaldab defektide modelleerimiseks nii *osalist käskude täitmist* kui ka nn. “uute” käskude tekkimist, mis pole ette nähtud mikroprotsessori käsusüsteemiga.

Rikked andmete säilitamise mehhanismis

Kuna registreid võib vaadelda kui ühte *mälumassiivi*, siis seda tüüpi rikete jaoks võib kasutada traditsioonilisi *mäluseadmete rikete mudeleid* [Goo 91,Abr 95, Mou 00]. Vead andmete säilitamisel mälus võivad toimuda järgmistel viisidel (andmete säilitamise mehhanismi mudel):

F9: ühes või enamas mäluelemendis võivad olla rikked konstant 1 või konstant 0;

F10: ühes või enamas mäluelemendis võivad tekkida vead 0→1 või 1→0 andmete edastamisel;

F11: kaks või rohkem mäluelemendi paari võivad olla seotud, s.t. kui ühes mäluelemendis olek muutub näiteks viisil $x \rightarrow y$, siis temaga paaris olevas mäluelemendis samuti olek muutub kas $x \rightarrow y$ või $y \rightarrow x$, kusjuures $x,y \in \{0,1\}$ ja $y \neq x$.

On näidatud [Tha 77], et rikked *mäludekooderis* ja ka sisend/väljundi registris on interpreteeritavad vigadena mälu massiivis.

Rikked andmete edastamise mehhanismis

Andmete edastamise funktsioon hõlmab kogu andmesiiret magistraalide kaudu registritest registritesse, kas otse või läbi andmetöötluse mehhanismi. Andmete edastamise mehhanismi rikete mudel koosneb järgmistest riketest:

F12: ühel või enamal signaalijuhtmel võivad olla rikked konstant 1 või konstant 0;

F13: üks või enam signaalijuhet võivad olla lühises ja vastavalt lühisrikke mudelile (JA-tüüpi või VÕI-tüüpi lühistele) võivad realiseerida vastavalt, JA- või VÕI-funktsioone lühises olevate juhtmete peal;

Rikked andmete töötlemise mehhanismis

Andmete töötlemise mehhanismi puhul spetsiaalseid hästi töötavaid rikkemudeleid pole välja pakutud. Ainsaks mõistlikuks viisiks oleks kasutada siin *hierarhilist lähenemisviisi*, mille puhul mehhanismi kuuluvad funktsionaalsed plokid esitatakse mingil madalamal tasandil, näiteks ventiilskeemide tasandil ja genereeritakse lokaalsed testid sellel samal madalal tasandil. Neid lokaalseid teste kasutatakse lõplike *testprogrammide* koostamisel taas kõrgtasandil.

Ehkki käesolevas peatükis kirjeldatud rikete mudel on laialt levinud testide genereerimiseks mikroprotsessoritele, on selle mudeli puuduseks liigne spetsialiseeritus, mis raskendab testprogrammide sünteesi

automatiseerimist. Samuti on mudel liialt orienteeritud kitsale mikroprotsessorite klassile, mistõttu ta pole kasutatav digitaalsüsteemidele üldjuhul.

2.4.2. Registersiirete tasandi rikete mudelid

Registersiirete rikete mudel on üles ehitatud eesmärgil kajastada teatavat hulka eri tüüpi funktsioneerimise anomaaliatena esitatavaid *funktsionaalseid rikkeid*. Rikete hulk on koostatud registersiirete tasandit kirjeldavate keelte *lauseanalüüsi* tulemusena.

Registersiirete tasandit kirjeldavas keeles esitatav lause omab üldjuhul järgmist struktuuri [She 88]:

$$K\text{: } (T,C)\ R_d \leftarrow f(R_{S1}, R_{S2},\ldots, R_{Sn}),\ \rightarrow N, \qquad (2\text{-}5)$$

kus

- K - on *märgend*,
- T - on ajaliselt määratud tingimus (*ajatingimus*), mis on tarvilik lausega määratud tegevuste realiseerimiseks;
- C - on *loogikatingimus*, mis on tarvilik lause (tegevuse) realiseerumiseks;
- R_d - on *sihtregister*, kuhu kirjutatakse tegevuse tulemus;
- R_{Si} - on *andmeallikas* (register, andmesisend);
- f - on *tehe*, mis sooritatakse andmeallikatest R_{Si} saadud operandidega;
- $\leftarrow$ - tähistab *andmesiiret* (tehte tulemuse kirjutamist sihtregistrisse R_d);
- $\rightarrow N$ - tähistab siiret järgmise lause (tegevuse) juurde aadressiga N.

Kasutades toodud tähistusi võime eristada 9 tüüpi *funktsionaalseid rikkeid*, mida kasutatakse registersiirete tasandil digitaalsüsteemide diagnostilisel modelleerimisel:

RT1: märgenditega seotud rike (K/K'), mis tähendab, et antud tegevus realiseeritakse vaid siis, kui järgmise tegevuse valik toimub suunamisega märgendi K' poole;

RT2: ajatingimuste ülesseadmisega seotud rikked (T/T'), kus ajatingimuse T asemel kontrollitakse hoopis ühte teist ajatingimust T';

RT3: loogikatingimuste ülesseadmisega seotud rikked (C/C'), kus tingimuse L asemel kontrollitakse hoopis ühte teist loogikatingimust L';

RT4: registrite (sihtregistrite R_d või andmeallikate $R_{S,i}$) dekodeerimisega seotud rikked (R_i/R_i'), kus registri R_i asemel valitakse hoopis teine register R_i';

RT5: tehete dekodeerimisega seotud rikked (f/f'), kus tehte f asemel realiseeritakse hoopis teine tehe f ';

RT6: juhtimisega seotud rikked, kus järgmine lause valitakse märgendi N', mitte aga märgendi N järgi;

RT7: rikked andmete salvestamisel või säilitamisel ((R_s)/(R_s)'), mis tähendab, et registri R_s sisu (R_s) võtab mingi defekti tõttu uue kuju (R_s)';

RT8: rikked andmete edastuse mehhanismis (←/←'), mis tähendab, et toimub mingi viga andmete edastamisel andmeallikast sihtkohta;

RT9: rikked andmete töötlemise mehhanismis ((f)/(f)', mis tähendab, et tehe f küll toimub aga tehte tulemus (f)'erineb oodatavast tulemusest (f).

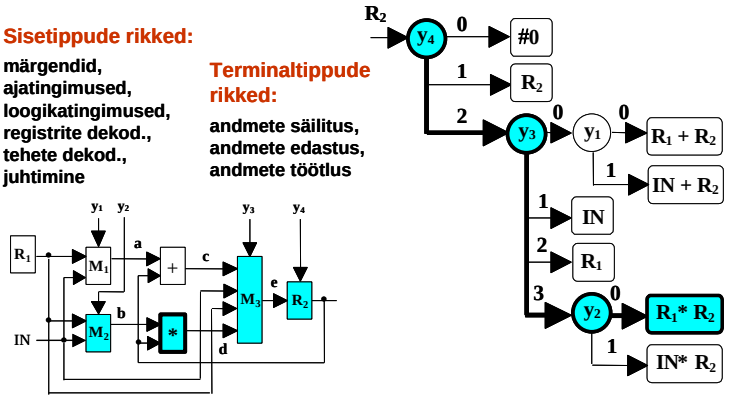
Kirjeldatud rikete mudel on üsna üldine ja sisukas, kuna igat digitaalsüsteemi on võimalik kirjeldada registertasandi keeltes selliste lausetega. Mudeli lihtsustamiseks on võimalik läbi viia ka rikete kollapsi protseduure toetudes rikete ekvivalentsuse ja dominantsuse suhetele.

Mudeli puuduseks on tema kohmakus – liiga palju erinevaid rikete tüüpe, mis kõik vajavad eraldi defineerimist ja eri käsitlemist.

2.4.3. Rikete modelleerimine otsustusdiagrammidega

Eespool kirjeldatud kõrgtasandi rikete mudelid on liigselt spetsialiseeritud ja on nõrgalt seotud süsteemi funktsioonide ning struktuuri mudeliga, s.t. et rikete mudel ei tulene otseselt süsteemi mudelist, vaid tuleb spetsiaalselt lisaks defineerida. Rikete mudelis kasutatavate rikete tüüpide rohkus teeb keeruliseks mudeliga töötamise. Tüüpide rohkuse tõttu on raske välja töötada ühtseid algoritme testide genereerimiseks ja rikete simuleerimiseks nii nagu see osutus võimalikuks loogikatasandil konstantse rikke mudelit kasutades.

Otsustusdiagrammide kasutamine annab paremad võimalused *universaalse rikkemudeli* väljatöötamiseks ka kõrgematel modelleerimise tasanditel. OD-mudel kajastab sisuliselt täpselt sama informatsiooni, mida esitatakse registersiirete tasandi traditsiooniliste keelte (näit. VHDL keele) abil, kuid palju kompaktsemal, korrastatumal ja diagnostilise modelleerimise seisukohast otstarbekamal kujul, kuna on selgelt välja toodud põhjus-tagajärg suhted. OD-mudelis on esitatavad kõik eelpool vaadeldud rikete mudelid, kusjuures neid ei tule lisaks süsteemi mudelile täiendavalt konstrueerida, vaid kõrgtasandi rikete mudel tuleneb otsustusdiagrammist endast otseselt (loomulikul viisil): iga tipuga on seotud üksainus rikete tüüp ja määratud hulk, kusjuures kõikide rikete käsitlus on standartne kõikide tippude jaoks. Praktilisest küljest lähtudes on aga siiski otstarbekam eristada OD sisetippude ja terminaaltippude käsitlust.



Joonis 2-27. *Kõrgtasandi rikete mudel otsustusdiagrammidel*

Vaatleme digitaalsüsteemi (operatsioonautomaadi andmeosa) ja selle otsustusdiagrammi joonisel 2-27. OD terminaltippudele vastavad andmete säilituse funktsioonid (tipud muutujatega R_1, R_2 ja *IN*), andmete edastuse funktsioonid (tipud muutujatega R_1, R_2 ja *IN*) ning andmete töötlemise funktsioonid (tipud avaldistega $R_1 + R_2$, $IN + R_2$, $R_1 * R_2$, $IN + R_2$), sisetippudele aga tingimuste kontrolli ja juhtimise funktsioonid. Joonisel 2-27 esitatud graafi sisetippudel kontrollitakse juhtsignaalide väärtusi. Üldjuhul võidakse otsustusdiagrammide sisetippudes kontrollida ka *siirdeaadresse* (märgendeid), aja- ning loogikatingimusi, ja *juhtsõnade* koodivälju, mille järgi adresseeritakse nii andmete allikaid (registreid, magistraale, sisendeid) kui ka selekteeritakse sooritatavaid tehteid.

Iga aktiveeritud tee otsustusdiagrammis juurtipust kuni terminaaltipuni kirjeldab digitaalsüsteemi ühte konkreetset *töörezhiimi*, mida saaks kirjeldada ka ühe lausega (2-5) mingis registersiirete tasandi kirjeldamise keeles. Seega kõik selle lause (2-5) komponendid on nii või teisiti esitatud ka otsustusdiagrammis. Kõik rikked, mis võivad mõjutada antud töörezhiimi teostust, on võimalik siduda OD tippudega vastaval aktiveeritud teel. Iga rike, mis on seotud konkreetse tipuga aktiveeritud teel, põhjustab kõrvalekalde sellelt teelt, mis tähendab, et jõutakse mingisse teise terminaaltippu, kus sooritatakse mingi teine tehe ettenähtud tehte asemel.

OD terminaltippudega, millistele vastavad andmetöötlusega seotud funktsioonid, on olukord pisut keerulisem. Nii nagu juba ka eelpool käsitletud mudelite puhul, on siin üldjuhul raske välja pakkuda hästi töötavat kõrgtaseme rikete mudelit. Ainsaks mõistlikuks meetodiks oleks läheneda probleemile hierarhiliselt ja sünteesida *lokaalsed testvektorid* terminaltippude testimiseks madalamal tasandil. Näiteks kui *tipuavaldiseks* on aritmeetiline liitmistehe, siis selle tehte testimiseks vajalikud operandid tuleks genereerida summaatori ventiiltaseme mudeli peal.

Universaalse rikkemudeli definitsioon

Igas digitaalsüsteemi kirjeldava otsustusdiagrammi tipus võivad aset leida järgmised rikked:

D1: tipust väljuv kaar on alati aktiveeritud (sellesse tippu jõudes jätkame teed graafil alati just selle kaare kaudu, sõltumata sellest, millist väärtust omab tipumuutuja);

D2: tipust väljuv kaar on katki ehk avatud (sellesse tippu jõudes katkeb edasine liikumine graafis, mistõttu ühtki tehet ei toimu);

D3: tipust väljutakse mingi teise kaare või korraga mitme teise kaare kaudu ettenähtud kaare asemel või sellele lisaks (viimasel juhul võidakse jõuda mitmesse terminaltippu korraga ning võidakse sooritada korraga mitu tehet, kusjuures saadud tehete tulemused võetakse kokku JA- või VÕI tehtega vastavuses süsteemis realiseeritud *tehnoloogiale*).

***Tabel 2-3**. Rikete mudelite vastavus*

DD-mudeli rikked	Registersiirete rikked	Mikroprotsessori rikked	
		Käsutasand	Mikrokäsutasand
D1	RT1 – RT6	F1	F1, F6
D2		F2, F4	F2, F4
D3		F3, F5	F3, F5, F7, F8
Terminaaltipp	RT7 – RT9	F9 - F13	F9 – F13

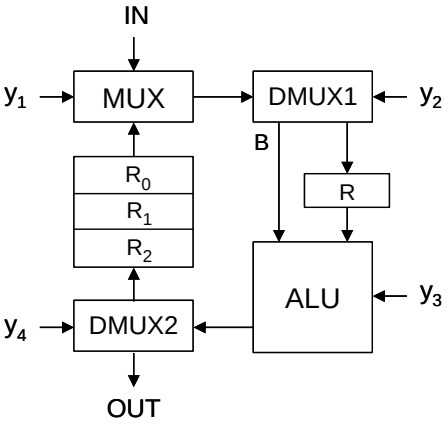
On kerge näha, et otsustusdiagrammide jaoks defineeritud ühtne rikkemudel katab eelpool vaadeldud mikroprotessori ja registersiirete tasandi rikkemudeleid ja on sisuliselt üldistuseks kõikidele eespool käsitletud kõrgtasandi rikete mudelitele.

Universaalse otsustusdiagrammi riketemudeli vastavus registersiirete tasandi ja mikroprotsessori rikete mudelitega on toodud tabelis 2-3.

Vaatleme järgnevalt mitmesuguste digitaalsüsteemide rikete modelleerimise näiteid otsustusdiagrammidel.

Registersiirete tasandi rikked

Joonisel 2-28 on esitatud digitaalsüsteemi *andmetöötlusplokk*, mille komponentide funktsioonid on toodud tabelis 2-4. Seade koosneb 4 registrist R, R_0, R_1, R_2, multipleksorist MUX, 2 demultipleksorist DMUX ning aritmeetikaseadmest ALU.



***Joonis 2-28**. Digitaalsüsteemi andmetöötlusplokk*

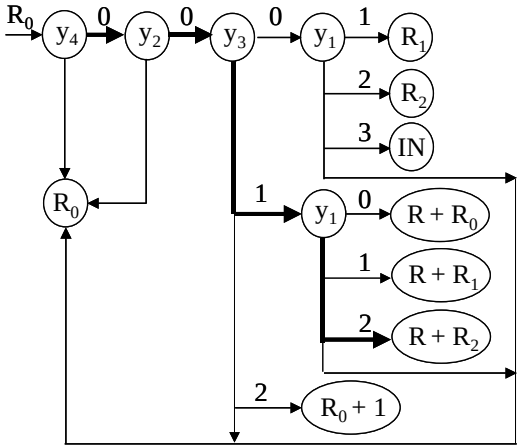
***Tabel 2-4**. Andmetöötlusploki funktsioonid*

MUX		DMUX1		ALU		DMUX2	
y_1	Tehe	y_2	Tehe	y_3	Tehe	y_4	Tehe
0	MUX = R_0	0	B = MUX	0	ALU = B	0	R_0 = ALU
1	MUX = R_1	1	R = MUX	1	ALU = B + R	1	R_1 = ALU
2	MUX = R_2	2	∅	2	ALU = B + 1	2	R_2 = ALU
3	∅	3		3	∅	3	OUT = ALU

Registri R_0 käitumine on esitatud otsustusdiagrammiga joonisel 2-29, kus argumentideks on juhtmuutujad (mikrokäsu operatsioonosa väljad) y_1, y_2, y_3, y_4, sisendmuutja *IN*, ning registermuutujad R, R_0, R_1, R_2.

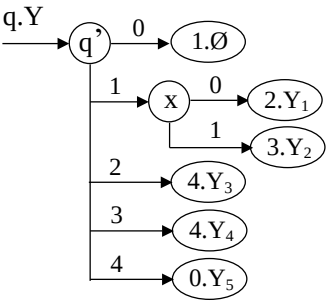
Registersiirete tasandi rikked RT2 – RT5 on otseselt kaetud OD-mudeli sisetippude riketega D1- D3, milliseid saab kergesti interpreteerida kui aja- või loogikatingimuste rikkeid, või registrite ning operatsioonide dekodeerimise (adresseerimise) rikkeid. Nii näiteks joonisel 2-29 tipp y_1 vastab registrite/andmeallikate dekooderile (MUX joonisel 2-28), tipp y_2 esitab loogikatingimust (demultiplekser DMUX2), tipp y_3 vastab mikrooperatsioonide dekodeerimisele (ALU), ning tipp y_4 vastab sihtregistri dekooderile (DMUX2).

Rikete klassid RT1 (märgendiga seotud rikked) ja RT6 (juhtimisega seotud rikked) esindavad juhtseadme rikkeid, mida samuti on kerge interpreteerida otsustusdiagrammidel.



Joonis 2-29. *Digitaalsüsteemi andmeosa otsustusdiagramm*

State	Logic condition	Next state	Micro-instruction
q’		q	
0		1	∅
1	x = 0	2	Y1
1	x = 1	3	Y2
2		4	Y3
3		4	Y4
4		0	Y5



Joonis 2-30. *Juhtseadme olekute/siirete tabel ja otsustusdiagramm*

Joonisel 2-30 on toodud juhtseadet kirjeldav *lõpliku automaadi* mudel (*olekute ning siirete tabel*) ning sellele vastav otsustusdiagramm. Viimane esitab vektorfunktsiooni $q.Y = \delta(x, q') . \lambda(x, q')$, kus x - on Boole’i muutuja (loogikatingimus), q – on arvutatav olekumuutuja, q' – on eelmisele ajahetkele vastav olekumuutuja ja Y – on juhtseadme

väljundmuutuja (mikrokäsk). Rikete klassidele RT1 ja RT6 vastavad graafi sisetipu *q*' ja terminaaltippude (*q.Y*) rikked. Tipu *x* rike otsustusdiagrammis vastab rikete klassile RT2 või RT3.

Rikete klasside RT7 – RT9 vastavus joonisel 2-29 toodud otsustusdiagrammi terminaaltippude riketega on toodud tabelis 2-5.

***Tabel 2-5**. Juhtseadme olekute/siirete tabel ja otsustusdiagramm*

Registersiirete tasandi rikked	Otsustusdiagrammi terminaaltippude rikked
RT7	R_0, R_1, R_2, R
RT8	R_0, R_1, R_2, R, IN
RT9	$R + R_0$, $R + R_1$, $R + R_2$, $R_0 + 1$

Mikroprotsessorite kõrgtasandi funktsionaalsed rikked

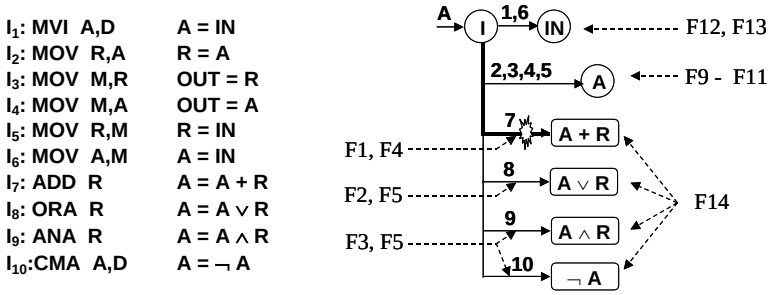
Joonisel 2-31 on toodud eelpool vaadeldud hüpoteetilise mikroprotsessori otsustusdiagrammi mudel koos võimalike rikete esitamisega vastavuses eelpool vaaadeldud rikete klassidega F1 – F13.

Oletame, et testitakse käsku I_7, mille tulemusena mudelis peaks aktiveeruma kaar $I = 7$ ja toimuma operatsioon $A = A + R$. Mudelis on illustreeritud järgnevaid rikkeid:

Rikete klasse F1 ja F4 esindab katkestus kaarel $I = 7$: operatsiooni $A = A + R$ ei toimu (D1). Rikete klassidele F2 ja F5 vastab mudelis viga, kus kaare $I = 7$ asemel aktiveerub mingi teine kaar, näiteks $I = 8$ ning realiseerub vale käsk I_8 : $A = A \vee R$ (D2). Rikete klasse F3 ja F5 esindab mudelis viga, kus kaare $I = 7$ asemel (või temaga koos) aktiveerub korraga mitu kaart ning realiseerub kombinatsioon kahest (või rohkemast) käsust I_9 : $A = A \wedge R$ ja I_{10} : $A = \neg A$ (D3). Kombinatsiooni tüüp (näiteks loogiline JA või VÕI) sõltub mikroprotsessori *tehnoloogilisest realisatsioonist* [Abr 86, Abr95].

Rikke klassidele F6, F7 ja F8, mis kajastavad vigu mikirokäskude tasandil, antud mudelil vasteid ei leidu, kuna tegemist on käsutasandiga. Juhul kui sama graaf joonisel 2-30 kajastaks mikrooperatsioone, vastaksid riketele F6, F7 ja F8, vastavalt rikked F1, F2 ja F3.

Rikete klassidele F9, F10 ja F11 vastavad vead mäluelementides või registrites, mida kajastavad mudelis terminaaltipud, kus märgenditeks on mälumuutujad, näiteks *A*. Rikete klassidele F12 ja F13 vastavad vead andmete edastamisel, näiteks sisendmagistraali *IN* kaudu, aga samuti ka registritest (näiteks tipp *A*). Rikete klassile F14 vastavad vead andmete töötlemisel (tipud $A + R$, $A \vee R$, $A \wedge R$, $\neg A$).



Joonis 2-31. *Mikroprotsessori kõrgtasandi rikete modelleerimine*

Otsustusdiagrammi tipurikke mudel võimaldab katta erinevaid rikete mudeleid süsteemi erinevatel abstraktsetel esitustasanditel. OD tipurikete füüsikaline sisu tuleneb sellest, millist konkreetset osa digitaalsüsteemist antud tipp modelleerib, milline sisuline tähendus on sellel tipul. Sõltuvalt digitaalsüsteemi või selle osa kirjeldamise detailsuse astmest otsustusdiagrammis sõltub ka tipurikke mudeli interpretatsioon ja detailsuse aste.

Näiteks SSBOD mudelis sisaldab tipu *m* rikkemudel kaks konstantset riket $\{x(m) \equiv 0$ ja $x(m) \equiv 1\}$, millele vastab üks ekvivalentsete rikete klass SSBOD-ga modelleeritavas ventiiltaseme skeemis. Seega SSBOD mudel katab täielikult klassikalise ventiilskeemidele vastava konstantse rikete mudeli.

Toetudes eelnevale, võime öelda, et OD tipurikke mudel on tegelikult klassikalise ventiiltaseme konstantse rikke mudeli üldistus digitaalsüsteemide kõrgtasandil esitatud kirjelduste jaoks. Kui *konstantse rikke mudel* toetub Boole'i algebrale ja on seetõttu kasutatav ainult loogikatasandil, siis OD tipurikke mudel on kasutatav nii loogikatasandil kui ka digitaalsüsteemide kõrgematel tasanditel.

2.5. Kokkuvõtteks

Käesolevas peatükis käsitlesime defektide ja rikete modelleerimise meetodeid digitaalsüsteemides. Kuna funktsionaalne lähenemisviis – testida süsteemi kõikvõimalikke funktsioone – osutub reaalsete objektide keerukuse tõttu ebapraktiliseks, siis struktuurne riketele põhinev lähenemisviis võimaldab saavutada küllaltki häid ja rahuldavaid tulemusi. Struktuurse lähenemisviisi puhul on diagnoosi objektiks rike, mitte funktsioon. Testi genereerimine tähendab leida iga võimaliku rikke jaoks teda avastav testvektor või -jada.

Eelpool toodud materjalist selgus, et kõigi võimalike rikete loendamine ja testimine on praktiliselt võimatu. Küll aga osutub võimalikuks testimise ülesannet lihtsustada, viies läbi kõigepealt rikete analüüsi, et välja valida ainult need rikked, mille testimine on oluline. Oluliste rikete välja selgitamiseks on vaja teada rikete tüüpe, on vaja osata klassifitseerida rikkeid, ning on vaja tunda rikete omadusi nagu liiasust, ekvivalentsust, maskeeruvust.

Keerukuse vastu võitlemiseks võib kasutada testide genereerimisel ka kõrgtasandi rikete mudeleid. Aga viimaste puhul pole alati selge, kui hästi nad esindavad reaalseid füüsikalisi defekte. Seetõttu, ehkki nad võimaldavad testide sünteesi oluliselt kiirendada, tuleb testide kvaliteedi kindlaks tegemiseks simuleerida saadud teste siiski ka madala tasandi rikete (defektide) hulgal, mistõttu testide sünteesi efektiivsus taas kahaneb.

Kõige tähtsamaks metoodiliseks võtteks võitluses keerukusega rikete käsitlemisel on nende hierarhiline modelleerimine funktsionaalse rikke mudeli abil. Nimetatud mudel võimaldab kujutada mitte üksnes ainult reaalseid defekte füüsikatasandilt loogikatasandile, vaid suvalise tasandi rikkeid järgmisele kõrgemale tasandile. Funktsionaalse rikke mudel on üldiseks aluseks hierarhilistele meetoditele nii testide sünteesil kui ka testide analüüsil.

Kirjandus

[Aba 89] M. Abadir et al. Logic Design Verification via Test Pattern Generation. *IEEE Trans. Comput-Aided Des. Integrated Circuits Syst.*, Vol. CAD-7, No 1, pp. 138-149.

[Abr 86] J.A.Abraham. Fault modeling in VLSI. *VLSI testing. North Holland* 1986, pp.1-27.

[Abr 95] M. Abramovici, M.A.Breuer, A.D.Friedman. Digital Systems Testing & Testable Designs. *Computer Science Press*, 1995, 653 p.

[Ake 59] S.B. Akers. On a Theory of Boolean Functions. *J. SIAM*, 7 (1959) H.4.

[Ake 78] S.B. Akers. Functional Testing with Binary Decision Diagrams. *J. of Design Automation and Fault-Tolerant Computing*, Vol.2, Oct. 1978, pp.311-331.

[Ben 98] B. Bennetts: Failures & Fault Models, Design for Testability Course Material. *Bennetts Associates*, February 1998.

[Boc 75] D. Bochmann. Einführung in die Strukturelle Automaten Theory. *VEB Verlag Technik Berlin*, 1975, 234 p.

[Boc 89] D. Bochmann, R.Ubar. Fehler in Automaten. *VEB Verlag Technik Berlin*, 1989, 216 p.

[Bry 86] Bryant R.E. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35 (8): 677-691, 1986.

[Bus 00] M.L.Bushnell, V.D.Agrawal. Essentials of Electronic testing. *Kluwer Acad. Publishers*, 2000, 690 p.

[Dre 98] R.Drechsler, B.Becker. Binary Decision Diagrams. *Kluwer Academic Publishers*, 1998, 200 p.

[Gho 91] S.Ghosh, T.J.Chakraborty. On Behavior Fault Modelling for Digital Designs. Kluwer Academic Publishers. *J. of Electronic testing: Theory and Applications*, 2, 1991, pp. 135-151.

[Gho 92] A. Ghosh et. al. Sequential Logic Testing and Verification. *Kluwer Academic Publishers*, 1992, 214 p.

[Goo 91] A.J. van de Goor. Testing Semiconductor Memories. Theory and Practice. *J. Wiley & Sons*, 1991. 512 p.

[Gri 99] R.P. Grimaldi. Discrete and Combinatorial Mathematics. *Addison-Wesley Longman*, Inc., 1999.

[Gup 85] A.K.Gupta, J.R.Armstrong. Functional Fault modelling and Simulation for VLSI Devices. *22nd Design Automation Conference*, 1985, pp.720-726.

[Hur 98] S.L.Hurst. VLSI Testing. Digital and Mixed Analog/Digital Techniques. *The Institution of Electrical Engineers*, Exeter, 1998, 532 p.

[Jha 90] N.K.Jha, S.Kundu. Testing and Reliable Design of CMOS Circuits. *Kluwer Academic Publishers*, 1990, 231 p.

[Jha 03] N.Lha, S.Gupta. Testing of Digital Systems. *Cambridge University Press*, 2003, 1000 p.

[Kun 97] W. Kunz, D.Stoffel. Reasoning in Boolean Networks. Logic Synthesis and Verification Using Testing Techniques. *Kluwer Academic Publishers*, 1997,230 p.

[Kär 96] R. Kärger. Diagnose von Computern. *B.G. Teubner, Stuttgart*, 1996, 416 p.

[Lal 97] P.K.Lala. Digital Circuit Testing and Testability. *Academic Press*, 1997, 199 p.

[Lee 59] C.Y.Lee. Representation of Switching Circuits by Binary Decision Programs. *The Bell System Technical Journal*, July 1959, pp.985-999.

[Lev98] R.Leveugle, R.Ubar. Synthesis of Decision Diagrams from Clock-Driven Multi-Process VHDL Descriptions for Test Generation. *Proc. of the 5th International Conference on Mixed Design of Integrated Circuits and Systems*. Lodz (Poland), June 18-20, 1998, pp. 353-358.

[Min 96] Minato S. Binary Decision Diagrams and Applications for VLSI CAD. *Kluwer Academic Publishers*, 1996, 141 p.

[Mol 04] P. Molitor, J.Mohnke. Equivalence Checking of Digital Circuits. Fundamentals, Principles, Methods. *Kluwer Academic Publishers*, 2004, 262 p.

[Mou 00] S.Mourad, Y.Zorian. Principles of Testing Electronic Systems. *J.Wiley & Sons, Inc*. New York, 2000, 420 p.

[Nic 03] N. Nicolici, B.M. Al-Hashimi. Power-Constrained Testing of VLSI Circuits. *Kluwer Academic Publishers*, 2003, 178 p.

[Nov 05] O.Novak, E.Gramatova, R.Ubar. Handbook of Electronic Testing. CTU Printhouse, Prague, 2005, 400 p.

[Pla 80] M.Plakk, R.Ubar. Digital Circuit Test Design using the Alternative Graph Model. Automation and Remote Control, Vol.41, No 5, part 2, Nov. 1980, *Plenum Publishing Corporation*, USA, pp. 714-722

[Pra 95] D.K. Pradhan. Fault-Tolerant Computer System Design. *Prentice Hall PTR*, New Jersey, 1995, 550 p.

[Rai 98] J.Raik, R.Ubar. Feasibility of Structurally Synthesized BDD Models for Test Generation. *Proc. of the IEEE European Test Workshop*, Barcelona (Spain), May 27-29, 1998, pp.145-146.

[Sac 98] M. Sachdev. Defect Oriented Testing for CMOS Analog and Digital Circuits. *Kluwer Academic Publishers*, 1998, 308 p.

[Sas 96] T. Sasao, M. Fujita. Representations of Discrete Functions. *Kluwer Academic Publishers*, 1996, 331 p.

[Sel 83] A.V.Seleznjov, B.T.Dobritsa, R.R.Ubar. Projektirovanije Avt. Sistem Kontrolja Bortovovo Oborudovanija Letatel'nyh Apparatov. *Mashinostrojenije*, Moskva, 1983, 224 p.

[She 88] L.Shen, S.Y.H.Su. A Functional Testing Method for Microprocessors. *IEEE Transactions on Computers*, Vol.37, No. 10, 1988, pp.1288-1293.

[Tha 71] A.Thaise. Boolean Differential Calculus. *Philips Res. Repts.*, vol. 26 (1971), pp.2229-246.

[Tha 77] S.M.Thatte, J.A.Abraham. Testing of Semiconductor Random Access Memories. *Proc. of 7th Int. Symp. on Fault-Tolerant Computing*, Los Angeles, June 1977, pp. 81-87.

[Tha 80] S.M.Thatte, J.A.Abraham. Test Generation for Micro-processors, *IEEE Trans. On Computers*, Vol. C-29, No. 6, pp.429-441, June 1980.

[Uba 76] R.Ubar. Test Generation for Digital Circuits with Alternative Graphs. *Proceedings of Tallinn Technical University No 409,* 1976, pp.75-81 (in Russian).

[Uba 80] R. Ubar. *Testovaja diagnostika tsifrovõh sistem.* I, II. *TPI, Tallinn,* 1981, 226 p.

[Uba 81] R.Ubar. Vektorielle Alternative Graphen und Fehlerdiagnose für digitale Systeme. *Nachrichtentechnik/Elektronik,* (31) 1981, H.1, pp.25-29.

[Uba 83] R.Ubar. Test Generation for Digital Systems on the Vector Alternative Graph Model. *Proc. of the 13th Annual Int. Symposium on Fault Tolerant Computing*, Milano, Italy, 1983, pp.374-377.

[Uba 96] R.Ubar. Test Synthesis with Alternative Graphs. *IEEE Design and Test of Computers.* Spring, 1996, pp.48-59.

[Uba 98] R.Ubar. Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams. *OPA (Overseas Publishers Assotiation) N.V. Gordon and Breach Publishers, Multiple Valued Logic*, Vol.4 pp. 141-157, 1998.

[Uba 05] O.Novak, E.Gramatova, R.Ubar. Handbook of Electronic Testing. Czech Technical University, Prague, 2005, 400 p.

Lühendid

ALU - Arithmetic Logic Unit 128
BOD - binaarne otsustusdiagramm 14
CMOS - Complementary Metal Oxide Semiconductor 87
DMUX - demultiplekser 128
DNK - disjunktiivne normaalkuju 18
D&T - disain ja test 14
ITS - isetestiv süsteem 14
KR - konstantrike 107
MUX - multiplekser 128
OD - otsustusdiagramm 14
SAF - Stuck-at Fault (konstantne rike) 94
SAF0 - Stuck-at-0 Fault (konstantne rike 0) 85
SAF1 - Stuck-at-1 Fault (konstantne rike 1) 85
SSBOD - struktuurselt sünteesitud binaarsed otsustusdiagrammid 39
TD - testikõlblik disain 14
VHDL - Very High Scaled Integrated Circuits Description Language 57
V_{DD} - Voltage Drain Drain 110
VSS - Voltage for Substrate & Sources 110

Indeks

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

Ü